PER IL PROGRAMMATORE MS-DOS E PS/2

SCHEDE GRAFICHE

TECNICHE AVANZATE DI PROGRAMMAZIONE

CONTIENE FLOPPY DISK MS-DOS

RICHARD WILTON

CONSIGLI, ESEMPI,
PROGRAMMI IN ASSEMBLER E IN C
PER SFRUTTARE TUTTE LE POSSIBILITÀ
DELLA VOSTRA SCHEDA GRAFICA

EGA-VGA-CGA-MDA-MCGA-HGC

GRUPPO EDITORIALE
JACKSON

PER IL PROGRAMMATORE MS-DOS E PS/2

SCHEDE GRAFICHE

TECNICHE AVANZATE DI PROGRAMMAZIONE

Microsoft PRESS ®

RICHARD WILTON

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

Titolo originale:
PROGRAMMER'S GUIDE TO PC & PS/2 VIDEO SYSTEMS

© Copyright per l'edizione originale:
1987 – RICHARD WILTON

© Copyright per la traduzione in lingua italiana:
Gruppo Editoriale Jackson S.p.A.

Tutti i diritti sona stati pubblicati in accordo con l'editore originale MICROSOFT PRESS, REDMOND, WASHINGTON

TRADUZIONE: Giovanni Cerami
IMPAGINAZIONE ELETTRONICA: Battistino Santini
REDATTORE DI COLLANA: Luigi Cerabolini
COPERTINA: Emiliano Bernasconi

Tutti i diritti sono riservati. Nessuna parte di questo libro può essere riprodotta, memorizzata in sistemi d'archivio, o trasmessa in qualsiasi forma o mezzo, elettronico, meccanico, fotocopia, registrazione o altri, senza la preventiva autorizzazione scritta dell'editore.

Gli autori e l'editore di questo volume si sono fatti carico della preparazione del libro e dei programmi in esso contenuti. Questa attività ha compreso la ricerca, lo sviluppo e il test di teorie e di programmi per determinare le loro funzionalità. Gli autori e l'editore non si assumono alcuna responsabilità, esplicita o implicita, riguardante questi programmi o il contenuto del testo.
Gli autori e l'editore non potranno in alcun caso essere ritenuti responsabili per incidenti o conseguenti danni che derivino o siano causati dall'uso dei programmi o dal loro funzionamento.

Indice

Ringraziamenti IV

Introduzione V

1 Hardware e firmware del video IBM 1

2 Programmazione dell'hardware 15

3 Modalità alfanumeriche 53

4 Modalità grafiche 99

5 Programmazione dei pixel 131

6 Linee 191

7 Cerchi ed ellissi 259

8 Riempimento di aree 283

9 Testo grafico 309

10 Set di caratteri alfanumerici 343

11 Blocchi di bit e animazione 395

12 Alcune tecniche avanzate di programmazione video 427

13 Subroutine grafiche in linguaggi ad alto livello 465

Appendice A: Riepilogo del BIOS video 495

Appendice B: Stampa dello schermo 565

Appendice C: Identificazione dei sistemi di visualizzazione 587

Glossario 601

Ringraziamenti

Il materiale contenuto nei Capitoli 6, 7 e 8 è stato in gran parte realizzato grazie agli sforzi iniziali di diversi esperti di grafica per computer. In ognuno di questi capitoli ho incluso dei riferimenti ad alcune delle loro più famose pubblicazioni. Se siete interessati agli algoritmi descritti in questi capitoli, cercate di procurarvi assolutamente le pubblicazioni originali e provatele personalmente.

Questo libro non avrebbe potuto essere scritto senza l'incoraggiamento della mia famiglia, dei miei amici e colleghi, che meritano un sentito ringraziamento per la loro pazienza ed il loro sostegno. La mia gratitudine va anche a Andy Fischer e a Charles Petzold, i quali hanno cortesemente revisionato parti del presente libro ed hanno offerto critiche e suggerimenti costruttivi.

Inoltre, naturalmente, uno speciale ringraziamento va a Claudette Moore, Jeff Hinsch e a molte altre persone della Microsoft Press che hanno pazientemente trasformato la bozza iniziale di questo libro nel prodotto finito.

Introduzione

Ricordo chiaramente il primo giorno in cui ho installato un nuovo adattatore grafico evoluto EGA dell'IBM nel mio PC IBM. Era bello avere un nuovo dispositivo video "evoluto", con migliore risoluzione e controllo sui colori e con funzioni non presenti in nessuno dei precedenti dispositivi video per PC dell'IBM. Ero pronto a quel punto a scrivere delle applicazioni grafiche veramente efficaci.

Per lo meno così pensavo. Il problema era che non riuscivo a capire come programmare il dispositivo. Non avevo documentazione tecnica (arrivò per posta sei mesi dopo, insieme ad un conto da pagare di 125 dollari). Tentai di disassemblare il BIOS ROM dell'EGA, ma lo studio di 6000 istruzioni in linguaggio macchina senza commenti sollevò più domande di quante ne riuscì a rispondere. Tentai disperatamente un approccio d'urto, modificando il contenuto delle locazioni di memoria e dei registri della macchina per vederne l'effetto, ma era come togliere dei pezzi da un'automobile per vedere se smette di funzionare.

Ciò che mi mancava erano i dettagli, le descrizioni concettuali del progetto hardware, le tabelle di descrizione dell'interfaccia programma e, soprattutto, esempi di codice sorgente di alcune tipiche tecniche di programmazione. Alcuni esempi selezionati di codice sorgente mi avrebbero risparmiato molte ore di tentativi e di frustrazione mentre tentavo di comprendere come programmare l'adattatore video.

Il presente libro ha tratto ispirazione dal doloroso ricordo di quell'esperienza. Il libro è pieno di esempi di codice sorgente. Il testo descrive il codice sorgente e viceversa. Questo libro contiene anche molte tabelle e descrizioni riassuntive dell'interfaccia di programmazione hardware. In breve, questo libro rappresenta quanto io desideravo avere quando iniziai a programmare il dispositivo video del PC.

Argomenti trattati

Il primo capitolo di questo libro contiene una panoramica generale sull'ambiente di visualizzazione. Esso descrive i dispositivi hardware video utilizzati generalmente su PC e su PS/2 che verranno trattati più approfonditamente negli altri capitoli del libro. Inoltre, esso rappresenta un'introduzione (nel caso non siate già degli esperti) alle note routine di supporto video BIOS ROM.

I successivi 10 capitoli contengono i dettagli della programmazione video dell'IBM. I primi capitoli trattano i principi fondamentali, tra cui l'architettura hardware, le modalità di visualizzazione e la natura dell'interfaccia tra i vostri programmi e l'hardware. Gli ultimi capitoli si estendono a partire dai principi fondamentali per dimostrare diverse tecniche per la produzione di output di testo e di grafica.

Gli ultimi due capitoli di questo libro introducono ai linguaggi di programmazione a basso e ad alto livello. Il Capitolo 12 è il capitolo dei fanatici dell'hardware: se amate lavorare con interrupt verticali o giocare con piani di bit, questo è il vostro capitolo. Infine,

il Capitolo 13 illustra come collegare i driver di dispositivo di visualizzazione a programmi scritti in linguaggi ad alto livello ed introduce diversi pacchetti commerciali di output su video.

A chi è diretto questo libro

Questo libro non è indirizzato specificatamente ai principianti. Ciò non significa che un programmatore che sta imparando come scrivere un codice operativo non potrà avvantaggiarsi di quanto spiegato. Al contrario, gli svariati esempi funzionali degli utili codici sorgente saranno preziosi per chiunque intenda realizzare della seria programmazione per i PC e per i PS/2. Ciononostante, maggiore è la vostra esperienza di programmazione, più strumenti avrete per risolvere i diversi e difficili problemi insiti nella programmazione video.

Linguaggi

Ho utilizzato il linguaggio Assembler e il C per la maggior parte degli esempi di programmazione contenuti in questo libro, sebbene abbia intenzionalmente evitato alcune delle strutture sintatticamente più criptiche del C. Se avete dimestichezza con l'Assembler e con un linguaggio ad alto livello come il C, il Pascal, il FORTRAN, il PL/1 o il BASIC strutturato, non dovreste avere problemi nella lettura degli esempi di codice sorgente.

Inoltre, il Capitolo 13 tratta le interfacce per diversi linguaggi ad alto livello utilizzando diversi modelli di memoria e protocolli di richiamo subroutine. Potete seguire le indicazioni contenute in quel capitolo per convertire qualsiasi esempio in codice sorgente richiamabile in C nel protocollo di richiamo subroutine utilizzato dal vostro interprete di linguaggio preferito.

Potreste voler utilizzare qualche altro strumento di programmazione se pensate di sperimentare gli esempi di codice sorgente che seguono. Ad esempio, un buon debugger di linguaggio Assembler potrebbe rivelarsi estremamente utile. Avrete probabilmente bisogno di un linker di oggetti se pensate di richiamare le routine in linguaggio Assembler riportate in questo libro da programmi scritti in linguaggi ad alto livello. Inoltre, mano a mano che i file sorgente e i moduli oggetto proliferano, potreste trovare piuttosto utile un programma di utilità *make* tipo UNIX per tenere tutto in ordine.

Sistema operativo

All'interno del presente libro si presuppone che si operi sotto MS-DOS, o PC-DOS, versione 2.0 o successiva. Tuttavia, non c'è nulla all'interno dei programmi in codice sorgente che verifichi il sistema operativo in uso, di conseguenza prestate attenzione nel trasferire il codice a versioni precedenti dell'MS-DOS o ad altri sistemi operativi.

Hardware

L'essenziale è avere un PC o un PS/2 collegato ad un video. La programmazione video è come il nuoto: una cosa è leggerne al riguardo, ma è tutt'altra cosa farne esperienza provando da soli. In effetti, se intendete effettuare moltissima programmazione video,

dovreste prendere in considerazione l'installazione di due diversi sottosistemi di visualizzazione e di video nel vostro PC. Con due serie separate di hardware di visualizzazione collegate allo stesso computer, potete eseguire un programma di debug (ricerca e correzione errori) su uno schermo mentre un programma di test produce un output sull'altro schermo. Questa configurazione hardware a doppio video permette di risparmiare parecchio tempo, in particolare quando state sviluppando routine di grafica come quelle descritte nei capitoli dal 5 al 9.

Segue un elenco dei diversi computer e adattatori video che io ho impiegato per sviluppare le tecniche trattate in questo libro:

Computer

PC/XT IBM

PC/AT IBM

PS/2 IBM modello 30

PS/2 IBM modello 60

Adattatori

Adattatore di visualizzazione monocromatica IBM

Adattatore colore/grafica IBM

Adattatore grafico evoluto IBM

Adattatore di visualizzazione del PS/2 IBM

Scheda grafica Hercules

Scheda a colori Hercules

Scheda grafica Hercules Plus

Scheda InColor Hercules

Se utilizzerete uno di questi computer o uno di questi adattatori, o se impiegherete un clone compatibile, dovreste essere in grado di eseguire gli esempi di codice sorgente.

Manuali

Per programmare in modo efficiente l'hardware di visualizzazione del PC IBM, dovete conoscere lo scopo del progetto dell'hardware e come si presume che il software e il BIOS di sistema interagiscano con esso. Le informazioni di base si trovano nei manuali di riferimento tecnico dell'IBM per il PC, il PC/XT, il PC/AT e i PS/2 e nei manuali di riferimento tecnico delle opzioni e degli adattatori. La maggior parte dei produttori secondari dei dispositivi di visualizzazione del PC IBM forniscono anche dettagliate informazioni tecniche sul loro hardware.

Il materiale contenuto in questo libro ha lo scopo di completare quanto esposto nella documentazione tecnica dei produttori. Ho tentato di seguire, dove possibile, la terminologia e le descrizioni hardware dei produttori. In ogni caso, la documentazione dei produttori è a volte confusa. Se riscontrate delle discrepanze tra la documentazione ufficiale e questo libro, potete (spero) basarvi su questo libro come fonte di informazioni corrette.

Tuttavia, in un libro di queste dimensioni, avrò certamente commesso degli errori. Sono graditi commenti, critiche e suggerimenti.

Ho scoperto che scrivere buoni programmi di visualizzazione è un compito impegnativo, ma la ricompensa è particolarmente soddisfacente. Spero di condividere con i lettori la parte impegnativa e la parte di soddisfazioni derivanti da questo libro.

1
Hardware e firmware del video IBM

Hardware video del PC e del PS/2 IBM
Adattatore video monocromatico e adattatore colore/grafica IBM
Scheda grafica Hercules - Scheda grafica Plus Hercules
Adattatore grafico evoluto IBM - Scheda InColor Hercules
Matrice grafica multi-colore - Matrice video grafica integrata

Introduzione all'interfaccia del BIOS ROM
L'interrupt 10H - Area dati di visualizzazione
Accesso al BIOS del video da un linguaggio ad alto livello

I sistemi di visualizzazione dei microcomputer migliorano sempre più. Dalla presentazione del PC IBM nel 1981, la tecnologia di progettazione è migliorata e si è ampliato il mercato per più potenti dispositivi hardware di visualizzazione. Sia l'IBM sia i suoi concorrenti hanno reagito sviluppando adattatori e dispositivi di visualizzazione sempre più sofisticati, parallelamente al software di cui questi dispositivi sono corredati.

Questo capitolo fornisce una panoramica sull'evoluzione dell'hardware video del PC e del PS/2 IBM. Questa panoramica non è da considerarsi assolutamente completa, ma copre i dispositivi più diffusamente impiegati offerti dall'IBM e dalla Hercules. Il capitolo si conclude con un'introduzione al BIOS del video IBM, una serie di driver incorporati nella ROM di tutti i PC e i PS/2 IBM, che fornisce un'interfaccia di programmazione di base per le applicazioni video.

Hardware video del PC e del PS/2 IBM

Un semplice PC/XT o PC/AT IBM non contiene hardware video incorporato, di conseguenza dovrete scegliere ed installare da voi l'hardware per il video. In una configurazione tipica, il video o monitor è collegato, tramite un cavo a 9 fili, ad un adattatore video installato all'interno del PC. Un adattatore video tipico è costituito da una piastra a circuito stampato dotata di un connettore a 9 contatti a cui si collega il cavo del monitor e da un connettore a lato scheda da 2 per 31 che si inserisce in uno degli slot della piastra madre del PC. La Figura 1 mostra questi connettori, oltre ad alcuni dei circuiti integrati comuni a molti adattatori video IBM. La circuiteria dell'adattatore video genera i segnali che controllano quanto viene visualizzato sullo schermo del video.

Quando acquistate un PC IBM, dovete decidere quale adattatore video e quale monitor utilizzerete. Gli adattatori video utilizzati più diffusamente e per cui è stato scritto più software sono gli adattatori di visualizzazione monocromatica, l'adattatore colore/grafica e l'adattatore grafico evoluto dell'IBM, nonché la scheda grafica monocromatica prodotta dalla Hercules.

Per contro, tutti i computer della serie PS/2 dell'IBM sono dotati di un sottosistema video incorporato, di conseguenza l'acquisto di un adattatore video separato non è necessario. Il sottosistema video dei modelli 25 e 30 del PS/2 è denominato Multi-Color Graphics Array (matrice grafica multi-colore). Nei modelli 50, 60 e 80 del PS/2, il sottosistema video integrato è conosciuto con il nome di Video Graphics Array (matrice video grafica). Per i PC/XT, i PC/AT e per il modello 30 del PS/2 è anche disponibile come adattatore il sottosistema Video Graphics Array (matrice video grafica). Questo adattatore possiede essenzialmente le stesse funzioni hardware del sottosistema integrato del PS/2.

Adattatore video monocromatico e adattatore colore/grafica IBM

Quando il PC fu presentato nel 1981, l'IBM offriva due adattatori video: l'adattatore video monocromatico (MDA) e l'adattatore colore/grafica (CGA). L'MDA è stato progettato per essere impiegato con un video monocromatico (il video monocromatico dell'IBM) che visualizza 80 colonne e 25 righe di testo alfanumerico. Il CGA supporta sia un video RGB (un video con segnali di input separati per il rosso, il verde e il blu), sia un normale apparecchio televisivo (che impiega un segnale video composito). Il CGA, naturalmente, può visualizzare informazioni grafiche su base punto per punto oltre che testo alfanumerico.

Anche se l'MDA e il CGA possono visualizzare 25 righe di testo da 80 colonne, la maggior parte degli utenti trovano la visualizzazione monocromatica verde dell'MDA più riposante per gli occhi. Ciò accade perché il video monocromatico utilizzato con un MDA presenta una risoluzione significativamente più elevata di quella di un qualsiasi monitor utilizzato con il CGA. La sua risoluzione è di 720 punti di larghezza per 350 punti di altezza; la risoluzione massima di un video controllato da un CGA è di 640 punti di larghezza per 200 punti di altezza.

Entrambi gli adattatori visualizzano i caratteri all'interno di matrici rettangolari di punti. Un semplice calcolo mostra che ogni carattere è largo 9 punti e alto 14 punti su un video monocromatico e solo 8 per 8 punti su un video CGA. La maggiore risoluzione dell'MDA produce caratteri più nitidi e quindi di più facile lettura. Per questo motivo, la maggior parte degli utenti di PC che devono leggere del testo preferiscono un MDA ad un CGA.

D'altro canto, molti utenti di computer devono visualizzare diagrammi, schemi e altre informazioni grafiche oltre al testo alfanumerico. Inoltre, la visualizzazione dei colori sullo schermo è essenziale in molte applicazioni per computer. Dal momento che l'MDA può visualizzare solo testo monocromatico, gli utenti di PC che hanno bisogno di output grafico possono giungere al compromesso di utilizzare il CGA, con la sua capacità grafica a colori punto per punto, ma con una visualizzazione di testo meno nitida.

Perché allora non collegare semplicemente il video monocromatico a più alta risoluzione ad un adattatore colore/grafica per ottenere il massimo da entrambi i dispositivi? Sfortunatamente, i segnali video generati da un MDA sono incompatibili con quelli richiesti per il controllo di un monitor CGA e viceversa. Il collegamento non conforme di monitor e adattatore porta ad un cattivo funzionamento del monitor invece che ad una visualizzazione con migliore risoluzione.

Se avete bisogno di un testo nitido e leggibile oltre che di una grafica a colori, e potete permettervi dei dispositivi addizionali, potete installare sia l'MDA sia il CGA nello stesso PC. Potrete così utilizzare il video monocromatico (collegato all'MDA) per l'elaborazione del testo ed un video a colori RGB (controllato dal CGA) per la grafica a colori.

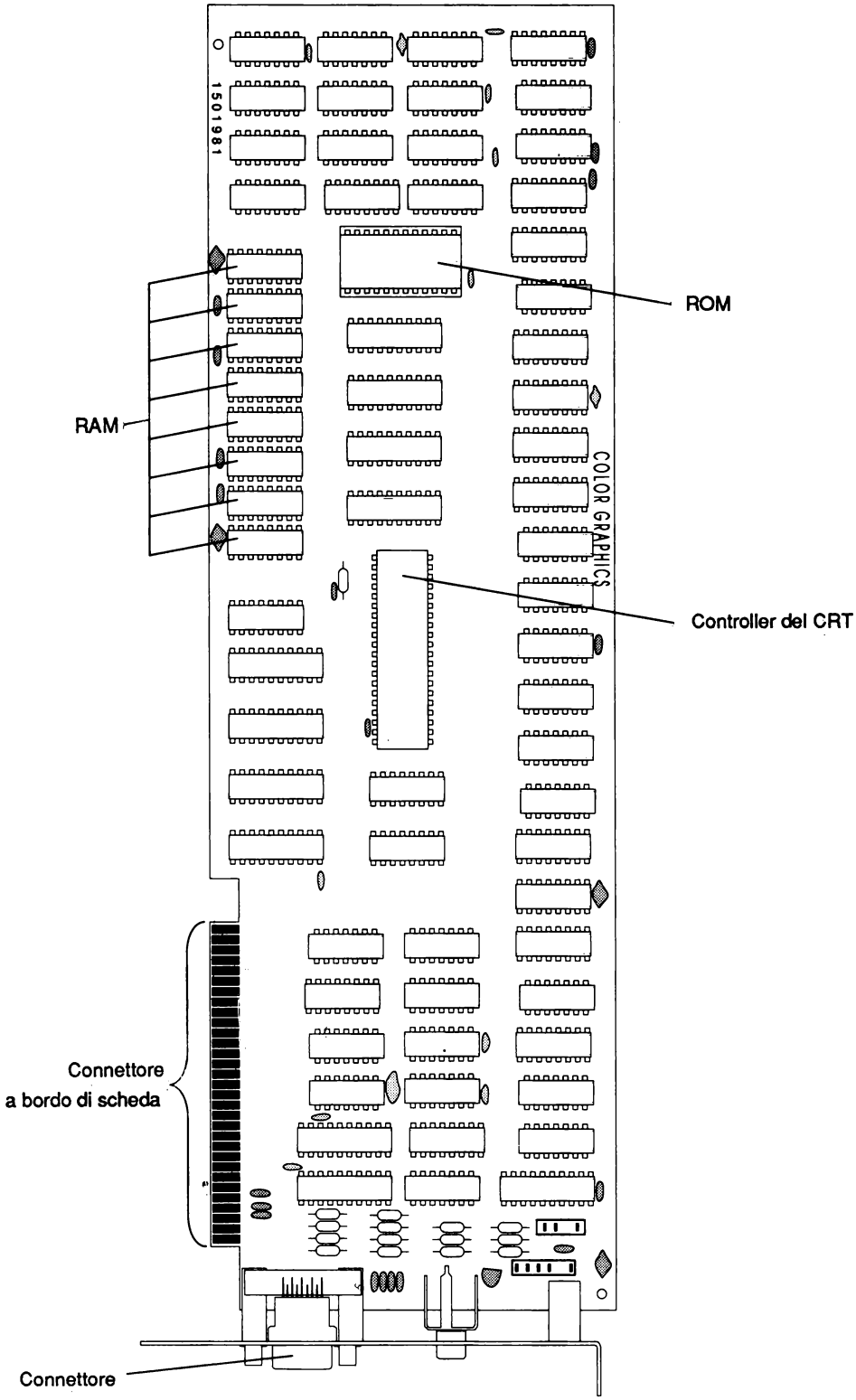


FIGURA 1. Un tipico adattatore video di un PC IBM

Scheda grafica Hercules

La soluzione Hercules al problema della visualizzazione di testo leggibile e di grafica punto per punto sullo stesso monitor è stata quella di aggiungere la capacità grafica ad un adattatore di video monocromatico. La scheda grafica monocromatica Hercules (HGC), presentata nel 1982, può visualizzare grafica e testo alfanumerico sullo stesso video verde monocromatico utilizzato con un MDA IBM (oltre alle proprie capacità grafiche, l'HGC duplica fedelmente la funzione dell'MDA originale dell'IBM). La capacità di visualizzare una combinazione di testo leggibile e di grafica monocromatica è sufficiente per molte applicazioni e quindi l'HGC rappresenta per molti utenti di PC un'opzione economica. Dal momento che riceve il supporto della maggior parte dei principali venditori di software, l'HGC si è assestata stabilmente nel mercato.

Scheda grafica Plus Hercules

L'HGC+ è stata presentata nel giugno del 1986. La grande differenza nell'aggiornamento dell'HGC originale risiedeva nel fatto che poteva visualizzare set di caratteri alfanumerici personalizzati basati su RAM, mentre l'MDA e l'HGC potevano visualizzare un solo set predefinito di caratteri alfanumerici basato su ROM. Dal momento che i caratteri alfanumerici possono essere visualizzati molto più rapidamente dei caratteri grafici punto per punto, l'uso dell'HGC+ può raddoppiare o triplicare la velocità di alcune applicazioni orientate al testo.

Adattatore grafico evoluto IBM

Una risposta diversa alla richiesta di una migliore risoluzione per testo e grafica è rappresentata dall'adattatore grafico evoluto (EGA) dell'IBM, presentato all'inizio del 1985. L'EGA può essere configurato in modo da emulare sia un MDA sia un CGA; ciò che rende "evoluto" l'EGA è il fatto che esso può eseguire funzioni non previste dai suoi predecessori. A differenza dell'MDA, l'EGA può produrre grafica punto per punto su un video monocromatico. Inoltre, l'EGA supera il CGA con la capacità di generare immagini grafiche o alfanumeriche a 16 colori con una risoluzione di 640 per 350 punti.

Sebbene le capacità di risoluzione e di colore dell'EGA non siano paragonabili a quelle del CGA, sia il testo sia la grafica appaiono molto più nitidi sull'EGA di quanto non avvenga sul CGA. La disponibilità di cloni EGA a basso costo e di applicazioni software di alta qualità che sfruttano le capacità dell'adattatore hanno reso l'EGA uno standard hardware "de facto" sul mercato.

Scheda InColor Hercules

La scheda InColor della Hercules, presentata nell'aprile del 1987, è essenzialmente una versione a 16 colori dell'HGC+. L'hardware della InColor emula perfettamente l'HGC+, quindi i programmi che operano correttamente sull'HGC+ possono operare senza modifiche sulla scheda InColor. La risoluzione della scheda InColor è identica a quella dell'HGC e dell'HGC+: 720 pixel orizzontali per 348 pixel verticali. Le capacità di colore dell'adattatore sono equivalenti a quelle dell'EGA. Questo adattatore può visualizzare 16 colori contemporaneamente a scelta da una tavolozza di 64 colori. L'adattatore deve essere utilizzato con un monitor a colori compatibile EGA che abbia una risoluzione di 350 righe verticali.

CONSIGLIO

Non confondete la scheda InColor con la scheda a colori Hercules, un clone perfezionato del CGA progettato per essere utilizzato nello stesso computer in cui è installata una HGC o una HGC+.

Matrice grafica multi-colore

La matrice grafica multi-colore (MCGA) è il sottosistema di visualizzazione integrato nei modelli 25 e 30 del PS/2. Dal punto di vista di un programmatore, l'MCGA assomiglia sotto molti aspetti al CGA, tuttavia l'MCGA presenta una risoluzione di gran lunga superiore (un massimo di 640 punti orizzontali per 480 verticali) e migliorate capacità di visualizzazione colore.

Una differenza significativa tra l'MCGA e gli adattatori citati precedentemente è rappresentata dal fatto che l'MCGA genera segnali video RGB analogici, mentre gli altri adattatori producono segnali RGB digitali. La differenza tra RGB analogico e digitale è simile alla differenza tra un interruttore a due posizioni ed un interruttore a potenziometro. Con i segnali RGB digitali, il video deve riconoscere solo se il segnale di un determinato colore (rosso, verde o blu) è on oppure off (acceso o spento). Nell'altro caso, un video che impiega segnali analogici RGB traduce il voltaggio di ogni segnale in un'ampia gamma di corrispondenti intensità di colore. Con l'MCGA può essere utilizzato solo un video analogico.

CONSIGLIO

Alcuni monitor possono essere configurati per segnali sia analogici sia digitali. Se utilizzate il cavo adatto, questi monitor possono essere collegati ad un MCGA se sono configurati per video analogici.

La ragione per impiegare un video analogico è che esso può visualizzare un'ampia gamma di colori. L'MCGA ha un convertitore di visualizzazione digitale-analogico (DAC) che permette al sottosistema di visualizzare fino a 256 diversi colori contempora-

neamente scelti da una tavolozza di 262.144 (256 K o 2^{18}) colori. Per l'impiego con l'MCGA, l'IBM fornisce un monitor analogico monocromatico, oltre ad un video a colori analogico. Con un monitor monocromatico, l'MCGA può visualizzare fino 64 gradazioni di grigio.

Matrice video grafica

Il termine "matrice video grafica" (VGA) si riferisce specificamente ad una parte della circuiteria del sottosistema di visualizzazione dei modelli 50, 60 e 80 del PS/2. La VGA è in effetti un singolo chip che integra la stessa serie di funzioni eseguite da diversi chip dell'EGA. Ciononostante, gli utenti generalmente utilizzano l'abbreviazione VGA per descrivere l'intero sottosistema di visualizzazione.

L'interfaccia di programmazione della VGA è analoga a quella dell'EGA, di conseguenza molti programmi scritti per l'EGA opereranno senza modifiche sulla VGA. La VGA è in grado di produrre in qualche modo una maggiore risoluzione (fino a 720 x 400 nelle modalità di testo, o 640 x 480 nelle modalità grafiche). Come l'MCGA, tuttavia, la VGA contiene un DAC di visualizzazione che può generare 256 colori contemporaneamente a scelta tra i 262.144 colori disponibili. Dal momento che la VGA genera gli stessi segnali RGB analogici dell'MCGA, deve essere utilizzata con gli stessi monitor a colori o monocromatici analogici.

Introduzione all'interfaccia del BIOS ROM

Nella ROM dei PC e dei PS/2 dell'IBM è incorporata una serie di routine BIOS (Basic Input/Output System, sistema di input/output di base). Le routine del BIOS ROM forniscono un'interfaccia alle funzioni hardware standard, tra cui l'orologio, la tastiera, i drive per disco rigido e per dischetti flessibili, e naturalmente il sottosistema di visualizzazione. Le routine BIOS di visualizzazione comprendono una serie di semplici strumenti per l'esecuzione di compiti fondamentali di programmazione video come ad esempio la scrittura di stringhe di caratteri sullo schermo, la cancellazione dello schermo, la modifica dei colori, eccetera.

Sebbene le routine BIOS ROM di visualizzazione siano talvolta lente e relativamente poco sofisticate, i programmi che le utilizzano sono trasportabili da un sottosistema di visualizzazione all'altro all'interno dei PC e dei PS/2 dell'IBM. Inoltre, la maggior parte dei produttori di cloni di PC IBM hanno duplicato le funzioni del BIOS dell'IBM nelle loro macchine. Di conseguenza, un programma che utilizza le routine BIOS per accedere all'hardware del video è probabile che sia più trasportabile di uno che non le utilizzi.

L'interrupt 10H

Le routine del BIOS sono scritte in linguaggio Assembler, quindi accedere ad esse è più semplice quando state realizzando dei programmi in linguaggio Assembler. A Tutte le routine di visualizzazione del BIOS si accede eseguendo l'interrupt 10H software 80x86 (il termine 80x86 si riferisce ai microprocessori della famiglia 8086 dell'Intel: 8086, 8088, 80286 e 80386). Per questo motivo, l'interfaccia di visualizzazione del BIOS ROM è conosciuta come interfaccia INT 10H. Il BIOS ROM supporta diverse funzioni di input/output video, a cui è possibile accedere individualmente eseguendo l'interrupt 10H. Le funzioni sono numerate; prima di eseguire l'interrupt 10H, si introduce il numero della funzione desiderata nel registro AH 80x86.

Al momento dell'esecuzione dell'interrupt, i restanti registri 80x86 di solito contengono parametri da passare alle routine del BIOS. Se la funzione INT 10H ritorna i dati al vostro programma, effettua questa operazione lasciando i dati in uno o più registri 80x86. Questo protocollo di passaggio parametri basato sui registri è stato progettato per l'impiego nei programmi in linguaggio Assembler.

Per vedere come viene utilizzata di solito l'interfaccia INT 10H, esaminiamo la routine in linguaggio Assembler *SetVmode()* del Listato 1. Questa routine può essere collegata ad un programma scritto nel linguaggio C della Microsoft (il trattino di sottolineatura che precede il nome della procedura, la parola chiave *near* della dichiarazione PROC e l'uso dello stack per il passaggio dei parametri seguono le convenzioni del C della Microsoft). Il fulcro della routine è rappresentato dalla chiamata al BIOS ROM per configurare l'hardware del video per una particolare modalità di visualizzazione (i dettagli di questa operazione sono trattati nel Capitolo 2 e nell'Appendice A).

```
                TITLE   'Listato 1'
                NAME    SetVmode
                PAGE    55,132

;
; Nome:         SetVmode
;
; Funzione:     Richiama BIOS ROM IBM per impostare una modalità video.
;
; Chiamante:    Microsoft C:
;
;                       annulla SetVmode(n);
;
;                       int n;                  /* modalità video */
;

ARGn            EQU     byte ptr [bp+4]; indirizzamento cornice stack

EQUIP_FLAG      EQU     byte ptr ds:[10h]; (nell'area di visualizzazione dati)

CGAbits         EQU     00100000b       ; bit per EQUIP_FLAG
MDAbits         EQU     00110000b
```

```
_TEXT          SEGMENTbyte public 'CODE'
               ASSUME cs:_TEXT

               PUBLIC _SetVmode
_SetVmode      PROC   near

               push   bp          ; mantiene registri del chiamante
               mov    bp,sp
               push   ds

               mov    ax,40h
               mov    ds,ax       ; DS - Area dati del video

               mov    bl,CGAbits  ; BL := bit che indicano la presenza del
                                  ;  CGA

               mov    al,ARGn     ; AL := numero modalità video desiderata

               mov    ah,al       ; verifica se modalità desiderata è
                                  ;  monocromatica
               and    ah,7cmp     ah,7
               jne    L01         ; salta se modalità desiderata non è 7 o 0Fh

               mov    bl,MDAbits  ; BL := bit che indicano la presenza
                                  ;  dell'MDA

L01:           and    EQUIP_FLAG,11001111b
               or     EQUIP_FLAG,bl   ; imposta bit in EQUIP_FLAG

               xor    ah,ah       ; AH := 0 (numero di funzione INT 10h)

               push   bp
               int    10h         ; richiama BIOS ROM per impostare modalità
                                  ;  video
               pop    bp

               pop    ds          ; ripristina registri del chiamante e
                                  ;  ritorna
               mov    sp,bp
               pop    bp
               ret

_SetVmode      ENDP

_TEXT          ENDS

               END
```

Listato 1. *Set V mode()*

La chiamata effettiva al BIOS del video è semplice. Per prima cosa, il numero di funzione desiderata viene introdotto nel registro AH *(XOR AH,AH)*. Successivamente, dopo aver conservato il contenuto del registro BP sullo stack *(PUSH BP)*, la routine richiama la funzione del BIOS ROM eseguendo l'interrupt 10H *(INT 10H)*.

Nel Listato 2, una routine complementare denominata *GetVmode()* richiede al BIOS il numero della modalità di visualizzazione corrente. La routine ottiene questo numero eseguendo la funzione 0FH dell'interrupt 10H. La funzione del BIOS ROM lascia il numero di modalità nel registro AL. *GetVmode()* ritorna quindi il numero al programma chiamante.

```
                TITLE   'Listato 2'
                NAME    GetVmode
                PAGE    55,132

;
; Nome:         GetVmode
;
; Funzione:     Richiama BIOS ROM IBM per impostare una modalità video.
;
; Chiamante:    Microsoft C:
;
;                       int         GetVmode();
;

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC _GetVmode
_GetVmode       PROC    near

                push    bp          ; conserva registri del chiamante
                mov     bp,sp

                mov     ah,0Fh      ; AH := 0Fh (numero funzione di INT 10h)

                push    bp
                int     10h         ; chiama BIOS ROM per avere numero
                                    ;  di modalità video
                pop     bp

                xor     ah,ah       ; AX := numero di modalità video

                mov     sp,bp
                pop     bp
                ret

_GetVmode       ENDP

_TEXT           ENDS

                END
```

Listato 2. *GetVmode()*.

Area dati di visualizzazione

Il codice che precede l'effettiva chiamata al BIOS ROM nel Listato 1 modifica una delle diverse variabili globali che riflettono lo stato del sottosistema di visualizzazione del PC. Queste variabili vengono aggiornate e a loro fanno riferimento tutte le routine video del BIOS ROM. Esse sono raccolte in un blocco di RAM chiamato, nella documentazione tecnica dell'IBM, area dati di visualizzazione (o area dati di controllo video). L'area dati di visualizzazione consiste di due blocchi di RAM. Il primo blocco si trova tra le locazioni di memoria 0040:0049 e 0040:0066, il secondo tra 0040:0084 e 0040:008A. Alcune routine BIOS del video fanno anche riferimento al campo di 2 bit in una variabile globale alla locazione 0040:0010 (chiamata nella documentazione tecnica dell'IBM EQUIP_FLAG). I bit 4 e 5 di questa variabile indicano una modalità video di default da utilizzarsi quando il computer viene caricato per la prima volta. Il codice di *SetVmode()* aggiorna questo campo di bit in conformità alla modalità video selezionata. Ad esempio, se per la modalità video desiderata viene richiesto un adattatore video monocromatico (MDA), il campo del bit di EQUIP_FLAG viene aggiornato di conseguenza (anche in questo caso, i dettagli sulle modalità video del BIOS ROM si trovano nel Capitolo 2 e nell'Appendice A).

CONSIGLIO

Nel corso del libro si trovano riferimenti all'interfaccia INT 10H, all'area dati di visualizzazione del BIOS e ai nomi simbolici di locazioni specifiche nell'area dati di visualizzazione che rivestono un interesse particolare. Se non avete già una certa dimestichezza con le funzioni INT 10H disponibili e con il contenuto dell'area dati di visualizzazione, un'attenta lettura dell'Appendice A potrebbe rivelarsi estremamente utile.

Accesso al BIOS del video da un linguaggio ad alto livello

Potete rendere accessibili le routine del BIOS ROM contenute in programmi scritti con linguaggi ad alto livello con una routine in linguaggio Assembler come *SetVmode()* o *GetVmode()*. I Listati 3 e 4 sono dei brevi pogrammi in C che possono essere eseguiti come comandi MS-DOS. Il programma del Listato 3 richiama *SetVmode()* per selezionare una modalità video. Questo programma può essere eseguito interattivamente o da un file batch. Il programma del Listato 4 richiama *GetVmode()* e ritorna il numero di modalità video in modo che possa essere utilizzato in un file batch (cioè, con comandi *IF ERRORLEVEL* ==).

```
/* Listato 3 */

main( argc, argv )
int             argc;
char            **argv;
{
                int     ModeNumber;
                void    SetVmode();


                if (argc != 2)          /* verifica sintassi riga comando */
                {
                  printf( "\nSintassi:  SETVMODE n\n" );
                  exit( 1 );
                }

                sscanf( argv[1], "%x", &ModeNumber );      /* ottiene numero
                                                              modalità
                                                              desiderata */

                SetVmode( ModeNumber );/* chiama BIOS ROM tramite INT 10h */
}
```

Listato 3. *Un programma in C basato su SetVmode().*

```
/* Listato 4 */

main()
{
                int     GetVmode();

                return( GetVmode() );
}
```

Listato 4. *Un programma in C basato su GetVmode().*

Il procedimento generale della produzione di un file eseguibile per uno di questi programmi consiste nel compilare il codice in C per produrre un modulo oggetto, nell'assemblare il codice in linguaggio Assembler per produrre un altro modulo oggetto e nel collegare i moduli oggetto per crere un file eseguibile. Se il codice sorgente in C del Listato 3 è contenuto in un file denominato *SM.C* ed il codice in Assembler del Listato 1 è stato salvato in *SETVMODE.ASM*, potete costruire il file eseguibile *SM.EXE* nel modo seguente:

```
msc             sm;                 (compila il codice C)
masm            setvmode;           (assembla la subroutine)
link            sm+setvmode;        (collega i moduli oggetto)
```

CONSIGLIO

Alcuni compilatori di linguaggi ad alto livello possono generare appropriati codici oggetto per il caricamento dei registri 80x86, eseguendo l'interrupt 10H e copiando il risultato dai registri al programma chiamante. Se il vostro compilatore ha questa capacità, potreste preferire accedere direttamente all'interfaccia INT 10H, invece di collegare una subroutine in linguaggio Assembler al vostro programma ad alto livello. Ad esempio, il Listato 5 usa la funzione *int 86()* del C della Microsoft per implementare *GetVmode()*.

```
/* Listato 5 */

#include        "dos.h"

main()
{
                struct BYTEREGS regs;       /* BYTEREGS definito in dos.h */


                regs.ah = 0x0F;              /* AH=0x0F (numero funzione
                                                         BIOS ROM) */

                int86( 0x10, &regs, &regs );/* esegue interrupt 10h */

                return( (int)regs.al );
}
```

Listato 5. *Funzione int86() del C della Microsoft.*

Nel BIOS ROM sono disponibili molte altre funzioni INT 10H. Il vostro programma applicativo accede a queste funzioni caricando i registri appropriati ed eseguendo l'interrupt 10H. Sebbene sia necessario ammettere che il supporto di INT 10H per l'input/output su video non è proprio perfetto, esso viene ampiamente utilizzato nel software dei sistemi operativi (compreso l'MS-DOS) oltre che in moltissime applicazioni. Se desiderate scrivere programmi grafici e per video veramente funzionali, dovrete acquisire dimestichezza con le capacità e con i limiti dell'interfaccia INT 10H.

2
Programmazione dell'hardware

Componenti funzionali dei sottosistemi di visualizzazione dei PC e dei PS/2 IBM
Monitor - Buffer del video
Hardware di visualizzazione del colore e dei caratteri
Il controller del CRT

Il ciclo di ripristino dello schermo
Temporizzazione orizzontale - Temporizzazione verticale

Programmazione del controller del CRT
MDA - CGA - Adattatori Hercules - EGA - MCGA - VGA

Calcoli di base del CRTC
Clock del punto - Temporizzazione orizzontale
Temporizzazione verticale

Il registro di stato del CRT

Modalità di visualizzazione
Risoluzione - Colori - Organizzazione del buffer del video

Controllo hardware della modalità di visualizzazione
MDA - CGA e MCGA - HGC
HGC+ e scheda InColor - EGA e VGA
Supporto BIOS del video

Combinazioni di sottosistemi di visualizzazione
MDA - Hercules - CGA - EGA - MCGA - VGA

Questo capitolo descrive l'hardware del video dei PC e dei PS/2 IBM dal punto di vista di un programmatore. Il presente capitolo tratta i principi fondamentali: quali parti del sottosistema di visualizzazione del computer possono essere programmate, come un programma interagisce con l'hardware e come vengono eseguiti i calcoli per la modifica del formato della visualizzazione. Molte delle tecniche di programmazione contenute nei capitoli successivi si basano sulle informazioni fondamentali che vedremo in questo capitolo.

Lo scopo di questo capitolo è quello di smitizzare l'interfaccia di programmazione hardware. Dal momento che la maggior parte dei programmatori si basano sul BIOS di visualizzazione per eseguire la maggior parte della programmazione (se non tutta) a livello hardware nelle loro applicazioni, un alone di mistero circonda il modo in cui il software intergisce con l'hardware del video. Naturalmente, dopo che ne saprete di più, potreste desiderare che fosse rimasto un fatto misterioso, ma più ne saprete, più i vostri programmi saranno in grado di agire sull'hardware del video.

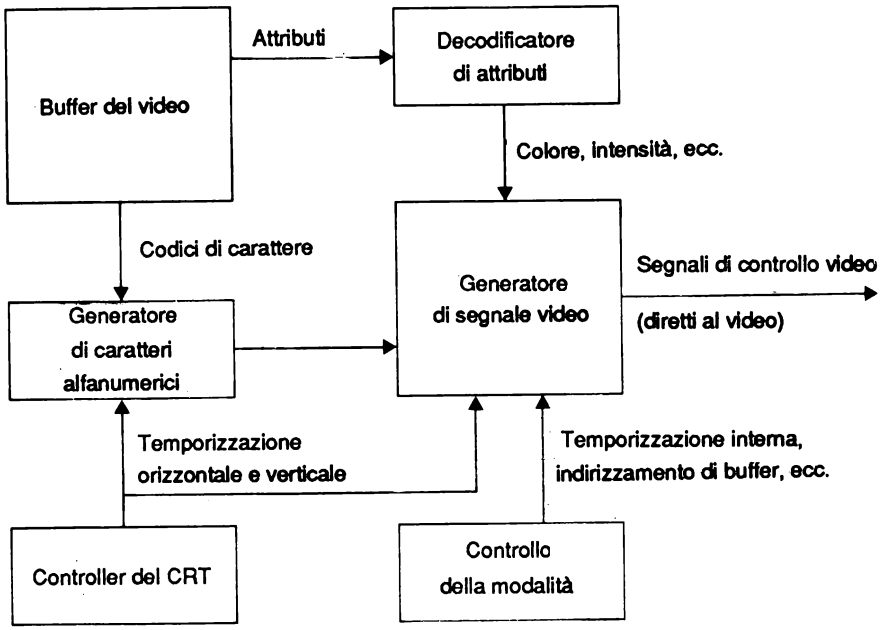


Figura 2. *Componenti programmabili (buffer del video, controller dell'attributo, eccetera) dei sottosistemi dei PC e dei PS/2 IBM. Alcuni a tutti questi componenti sono sotto controllo software in ogni sottosistema di visualizzazione descritto in questo libro*

Componenti funzionali dei sottosistemi di visualizzazione dei PC e dei PS/2 IBM

Quando scrivete dei programmi che interagiscono con l'hardware di visualizzazione IBM, è utile avere ben presente la relazione esistente tra i componenti programmabili dei sottosistemi di visualizzazione IBM (vedere la Figura 2). Non è necessario avere la comprensione dell'hardware di un progettista di circuiti per scrivere una buona interfaccia video. E' necessario conoscere dove e come il vostro programma può interagire con l'hardware per produrre efficientemente un output su video.

Monitor

Il componente più tangibile dell'hardware di visualizzazione di un computer è rappresentato dal monitor, o video. Tuttavia, non c'è nulla che possiate programmare direttamente nell'hardware del monitor. E' il sottosistema di visualizzazione del computer che contiene l'hardware programmabile. I segnali generati dal sottosistema di visualizzazione controllano ciò che appare sullo schermo.

Il monitor si differenzia da un apparecchio televisivo per uso domestico per il fatto che un gruppo di segnali separati di temporizzazione e di colore lo controllano. Per contro, un apparecchio televisivo decodifica un singolo segnale "composito" che contiene informazioni di temporizzazione, di colore e di suono. Sebbene alcuni adattatori video del PC IBM possano generare questo tipo di segnali di output composito, oltre che segnali di controllo diretto che vengono impiegati dai monitor per computer, la maggior parte degli utenti evitano di utilizzare un televisore con i loro computer. Sia il testo sia i colori appaiono più nitidi su un monitor per computer di quanto non appaiano su uno schermo televisivo composito.

Tutti i monitor trattati nel corso di questo libro sono dispositivi a scansione a reticolo. L'immagine sullo schermo di un monitor viene realizzata da un gruppo di linee orizzontali molto vicine denominato reticolo. Un raggio elettronico scorre ogni linea in successione da sinistra verso destra, a partire dall'angolo superiore sinistro dello schermo. Mano a mano che il raggio scorre ogni linea, il colore e la luminosità di diverse centinaia di punti (pixel) della linea vengono modificati e l'intero reticolo appare come un'unica immagine.

Concettualmente, potete considerare il raggio elettronico come se avesse "colore" e "intensità", ma nei monitor a colori il raggio in effetti è composto da tre raggi elettronici separati. Ogni raggio controlla la visualizzazione sullo schermo di uno dei tre colori primari (rosso, verde e blu). Ogni pixel di una visualizzazione a colori viene fisicamente rappresentato da una piccola triade di punti luminosi molto ravvicinati rossi, verdi e blu o strisce di materia fosforescente. I tre raggi elettronici sono mascherati in modo tale che ognuno illumini i punti di un solo colore primario. Di conseguenza, l'intensità relativa dei raggi mentre scorrono su ogni triade determina il colore e la luminosità dei pixel.

Naturalmente, a meno che non utilizziate una lente di ingrandimento o che non guardiate lo schermo molto da vicino, non riuscirete a percepire individualmente i punti rossi, verdi e blu ma piuttosto avrete la percezione di una mescolanza di colori.

Buffer del video

Il buffer del video è un blocco di RAM nel sottosistema di visualizzazione all'interno del quale vengono memorizzati i dati visualizzabili. Questa RAM si trova all'interno dello spazio indirizzabile della CPU del computer, quindi un programma può leggere e scrivere sul buffer del video nello stesso modo in cui accede ad una qualsiasi area di RAM.

La circuiteria di visualizzazione del sottosistema aggiorna, o ripristina, lo schermo leggendo ripetutamente e continuativamente i dati all'interno del buffer del video. Ogni bit o gruppo di bit del buffer del video specifica il colore e l'intensità di una particolare locazione sullo schermo. Lo schermo viene ripristinato dalle 50 alle 70 volte al secondo, in conformità al sottosistema di visualizzazione utilizzato. Ovviamente, quando un programma cambia il contenuto visualizzabile del buffer del video, lo schermo viene modificato quasi immediatamente.

La quantità effettiva di RAM disponibile come buffer del video varia a seconda del sottosistema di visualizzazione. La maggior parte dei sottosistemi di visualizzazione IBM incorporano buffer del video sufficientemente capienti da contenere più di uno schermo di dati visualizzabili, quindi, ogni volta, solo una parte del buffer è visualizzabile sullo schermo (il Capitolo 3 spiega come utilizzare completamente la RAM video disponibile).

Hardware di visualizzazione del colore e dei caratteri

Tutti i sottosistemi di visualizzazione IBM incorporano hardware che legge e decodifica i dati contenuti nel buffer del video. Ad esempio, un generatore di caratteri alfanumerici traduce i codici ASCII dal buffer del video in configurazioni di punti che compongono i caratteri sullo schermo. Un decodificatore di attributo traduce altri dati del buffer del video in segnali che producono colori, sottolineature e così via. Il software può controllare questi ed altri componenti specifici del sottosistema di visualizzazione; i prossimi capitoli descrivono in dettaglio questo tipo di programmazione.

Il controller del CRT

Il controller del CRT (CRTC) genera segnali di temporizzazione orizzontale e verticale. Esso incrementa anche un contatore di indirizzo di buffer del video ad una velocità sincronizzata con i segnali di temporizzazione. La circuiteria del video legge i dati dal buffer del video utilizzando il valore di indirizzo del CRTC, decodifica i dati ed invia i

segnali risultanti di colore e di luminosità al monitor insieme ai segnali di temporizzazione del CRTC. In questo modo il CRTC sincronizza la visualizzazione dei dati dal buffer del video con i segnali di temporizzazione che guidano la visualizzazione.

Il CRTC effettua diverse altre funzioni di varia natura. Tra queste, la determinazione della dimensione e della posizione di visualizzazione del cursore hardware, la selezione dell'area del buffer del video da visualizzare, la localizzazione del trattino di sottolineatura hardware e il rilevamento dei segnali della penna ottica (il Capitolo 3 contiene degli esempi della programmazione del CRTC per alcune di queste funzioni).

Su MDA, CGA e schede Hercules, il CRTC è costituito da un singolo chip, il 6845 della Motorola. Su EGA, il CRTC è costituito da un chip LSI (large-scale integration, integrazione su larga scala) personalizzato progettato dall'IBM. Su MCGA, il CRTC fa parte del proprio Memory Controller Gate Array. Il CRTC della VGA è un componente del chip singolo della matrice video grafica. A prescindere dalle diverse implementazioni hardware, il CRTC può essere programmato per generare una vasta gamma di parametri di temporizzazione in tutti questi sottosistemi. Prima di addentrarci nelle tecniche della programmazione del CRTC, però, è importante rivedere come i segnali di temporizzazione del CRTC controllino la visualizzazione del monitor di un'immagine di video a scansione a reticolo.

Il ciclo di ripristino dello schermo

L'immagine del video viene ripristinata in modo ciclico dalle 50 alle 70 volte al secondo, in conformità alla configurazione del sottosistema di visualizzazione. Nel corso di ogni ciclo di ripristino, il raggio elettronico passa sullo schermo procedendo a zigzag, a partire dalla sinistra della linea orizzontale superiore del reticolo (vedere Figura 3). Dopo aver scorso una linea da sinistra a destra, il raggio viene deviato verso il basso sull'inizio della linea successiva fino al completamento della scansione di tutto il reticolo. Successivamente, il raggio torna sull'angolo superiore sinistro dello schermo, ed il ciclo si ripete.

Temporizzazione orizzontale

Mentre il raggio si sposta sullo schermo, avvengono una serie di eventi temporizzati in modo preciso. All'inizio di ogni linea, il raggio elettronico viene acceso in risposta al segnale di abilitazione visualizzazione generato dal CRTC. Mentre il raggio scorre da sinistra a destra su una linea, la circuiteria del video utilizza il contatore di indirizzi del CRTC per leggere una sequenza di byte dal buffer del video. I dati vengono decodificati ed utilizzati per controllare i segnali di colore e di luminosità inviati al monitor. Mentre il raggio scorre sullo schermo, il suo colore e la sua luminosità variano in risposta a questi segnali.

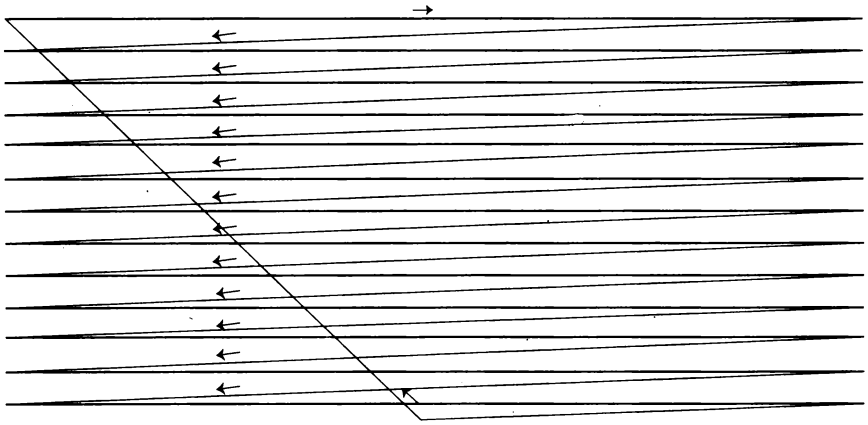


Figura 3. *Il percorso seguito dal raggio elettronico durante la scansione a reticolo.*

Vicino al bordo destro dello schermo, il CRTC disattiva il segnale di abilitazione visualizzazione e nessun dato viene visualizzato dal buffer del video. Il CRTC genera quindi un segnale di sincronizzazione orizzontale, che fa sì che il monitor devii il raggio elettronico verso sinistra e verso il basso, sull'inizio della linea orizzontale successiva del reticolo. A quel punto il CRTC riattiva il segnale di abilitazione visualizzazione per visualizzare la riga successiva di dati.

Il breve intervallo di tempo che intercorre tra la fine di una linea di dati visualizzati e l'inizio di quella successiva viene definito intervallo di soppressione orizzontale. Siccome l'intervallo di ritracciamento orizzontale (l'intervallo di tempo necessario per deviare il raggio sull'inizio della linea successiva) è più breve dell'intervallo di soppressione orizzontale, viene generata una serie di sovrascansioni orizzontali ad entrambe le estremità di ogni linea (vedere Figura 4).

Nel corso delle sovrascansioni orizzontali, il raggio elettronico può essere lasciato acceso, producendo così la visualizzazione di un colore di sovrascansione, o cornice. Tuttavia, il motivo principale per cui la sovrascansione orizzontale è prevista in un sottosistema di visualizzazione è per fornire un certo margine di errore nella centratura del reticolo, in modo che nessun dato venga perso sui bordi dello schermo.

Temporizzazione verticale

Una volta che il raggio elettronico ha scorso tutte le linee orizzontali del reticolo, il segnale di abilitazione visualizzazione viene disattivato. Il CRTC genera quindi un segnale di sincronizzazione verticale, che indica al monitor di deviare il raggio elettronico dalla parte inferiore dello schermo all'angolo superiore sinistro.

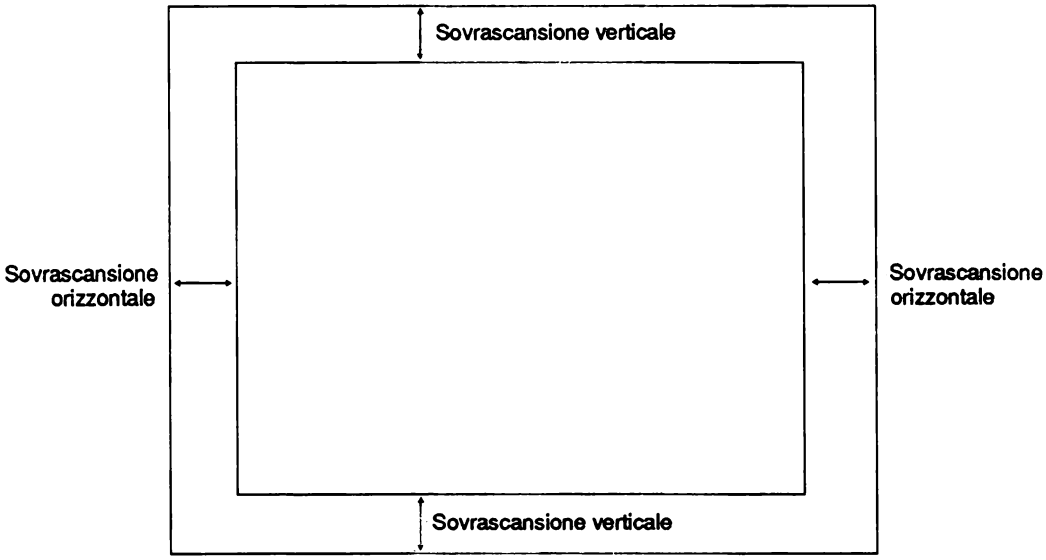


Figura 4. *Sovrascansione.*

L'intervallo di ritracciamento verticale (durante il quale il raggio si sposta dal bordo inferiore a quello superiore dello schermo) è più breve dell'intervallo di soppressione verticale (durante il quale non viene visualizzato alcun dato del buffer del video), quindi ci sono aree di sovrascansione verticale alle estremità superiore ed inferiore del reticolo (vedere Figura 4). Come nel caso della sovrascansione orizzontale, la sovrascansione verticale fornisce una cornice oltre che un margine di sicurezza in modo che il reticolo possa essere centrato sullo schermo.

Programmazione del controller del CRT

L'interfaccia di programmazione del CRTC è ben definita e di facile utilizzo. Lo stesso approccio generale di programmazione si applica a tutti i sottosistemi di visualizzazione dei PC e dei PS/2 IBM.

MDA

Il CRTC dell'adattatore video monocromatico, il 6845 Motorola, ha diciannove registri di dati interni a 8 bit. Il contenuto di ogni registro controlla varie caratteristiche dei segnali di temporizzazione generati dal 6845 (vedere Figura 5). Uno di questi registri è un registro di indirizzo; il suo contenuto indica a quale degli altri 18 registri è possibile accedere. La maggior parte dei registri sono di sola scrittura, ma i registri 0EH e 0FH,

che controllano la posizione del cursore hardware, possono essere sia letti sia scritti. Sull'MDA, il registro di indirizzo del 6845 è collegato (mappato) ad una porta di I/O a 3B4H e i restanti 18 registri sono tutti mappati alla porta di I/O successiva a 3B5H.

Per accedere ai registri dei dati del 6845, è necessario prima scrivere il numero di registro sul registro di indirizzo del 6845 (porta di I/O 3B4H). Successivamente, si accede al registro specificato dei dati con un'operazione di lettura o scrittura di I/O alla porta 3B5H. Ad esempio, il Listato 6 mostra come determinare la locazione corrente del cursore leggendo il contenuto dei registri 0EH e 0FH sul 6845. Questi due registri (locazione alta del cursore e locazione bassa del cursore) contengono i byte di ordine principale e secondario della locazione del cursore relativamente all'inizio del buffer del video.

Registro	Nome	Accesso Lettura/Scrittura
00H	Totale orizzontale	Solo scrittura
01H	Visualizzato orizzontalmente	Solo scrittura
02H	Posizione di sincronizzazione orizzontale	Solo scrittura
03H	Ampiezza impulso di sincronizzazione orizzontale	Solo scrittura
04H	Totale verticale	Solo scrittura
05H	Regolazione totale verticale	Solo scrittura
06H	Visualizzato verticalmente	Solo scrittura
07H	Posizione di sincronizzazione verticale	Solo scrittura
08H	Modalità interlace	Solo scrittura
09H	Linea di scansione massima	Solo scrittura
0AH	Inizio cursore	Solo scrittura
0BH	Fine cursore	Solo scrittura
0CH	Inizio indirizzo alto	Solo scrittura
0DH	Inizio indirizzo basso	Solo scrittura
0EH	Locazione alta cursore	Lettura/scrittura
0FH	Locazione bassa cursore	Lettura/scrittura
10H	Penna ottica alta	Solo lettura
11H	Penna ottica bassa	Solo lettura

Figura 5. *Registri dati del CRTC del 6845 Motorola (per adattatori video MDA, CGA e Hercules).*

```
mov     ax,40h
mov     es,ax        ; ES :=  segmento dati BIOS video
mov     dx,es:[63h]; DX := 3x4h (3B4h o 3D4h)

mov     al,0Eh
out     dx,al        ; seleziona registro locazione alta cursore

inc     dx
in      al,dx        ; legge registro selezionato a 3x5h
mov     ah,al        ; AH := byte alto della locazione cursore
dec     dx
```

```
            mov     al,0Fh
            out     dx,al     ; seleziona registro locazione bassa cursore

            inc     dx
            in      al,dx     ; AX := offset del cursore relativamente
                              ;  all'inizio del buffer del video

; converte a riga e colonna carattere

            mov     dx,es:[4Eh]          ; DX := CRT_START (offset inizio
                                         ;  buffer in byte)
            shr     dx,1                 ; converte in parole
            sub     ax,dx                ; sottrae da offset del cursore
            div     byte ptr es:[4h]     ; divide per CRT_COLS
            xchg    ah,al                ; AH := riga, AL := colonna
```

Listato 6. *Lettura dei registri della locazione del cursore del 6845.*

Con l'MDA non esistono praticamente motivi per modificare i valori dei registri del 6845 tranne che 0AH e 0BH (inizio cursore e fine cursore) e 0EH e 0FH (locazione alta e bassa cursore). I registri da 00H a 09H controllano i segnali di temporizzazione orizzontale e verticale, che non dovrebbero essere modificati. I registri 0CH e 0DH (inizio indirizzo alto e inizio indirizzo basso), che indicano quale parte di buffer del video dell'MDA è visualizzata, dovrebbero sempre essere impostati a 0.

CGA

Il CRTC dell'adattatore colore/grafica è un 6845 Motorola, come quello dell'MDA. La stessa tecnica di programmazione utilizzata per accedere al CRTC su MDA funziona anche su CGA. Su CGA, tuttavia, il registro di indirizzo del CRTC è collegato alla porta di I/O 3D4H e ai registri dei dati si accede a 3D5H. Se scrivete un programma che può essere eseguito sia su MDA sia su CGA, potete sfruttare il fatto che le routine BIOS del video in entrambe le famiglie di PC e di PS/2 tengono il valore della porta di I/O del registro di indirizzo del CRTC in una variabile. Molti degli esempi di programmazione riportati in questo libro fanno riferimento a questa variabile, ADDR_6845, che è posta a 0040:0063 nell'area dati di visualizzazione del BIOS.

Adattatori Hercules

Analogamente all'MDA e al CGA, la scheda grafica, la scheda grafica Plus e la scheda InColor Hercules utilizzano come CRTC un 6845 Motorola. I registri del CRTC sono collegati alle porte di I/O 3B4H e 3B5H su tutti gli adattatori Hercules. Sebbene la scheda InColor sia un adattatore a colori, essa utilizza la porta di I/O e gli indirizzi di buffer del video dell'MDA per poter mantenere la compatibilità con l'MDA e con gli adattatori monocromatici Hercules.

CONSIGLIO

Su tutti gli adattatori video Hercules (come su EGA, MCGA e VGA), potete impostare sia i registri di indirizzo sia quelli dei dati del CRTC con un'operazione di scrittura di porta a 16 bit (*OUT DX,AX*) invece che con due operazioni di scrittura di porta a 8 bit (*OUT DX,AL*). Ad esempio, le due sequenze di codice che seguono producono lo stesso effetto sul CRTC.

```
mov     dx,3B4h         ; registro d'indirizzo del CRTC
mov     al,0Ch          ; numero registro del CRTC
out     dx,al           ; seleziona questo registro
inc     dx              ; DX := 3B5h (registro dati del CRTC)
mov     al,8            ; dati
out     dx,al           ; memorizza dati nel registro
```

```
mov     dx,3B4h         ; registro d'indirizzo del CRTC
mov     ax,080Ch        ; AL := numero registro, AH := dati
out     dx,ax           ; memorizza dati nel registro
```

EGA

Il CRTC dell'adattatore grafico evoluto è un chip esclusivo LSI con una serie di registri diversi da quelli del 6845 (vedere Figura 6). L'interfaccia di programmazione è simile a quella del 6845, ma le assegnazioni e i formati di registro sono abbastanza diversi da far sì che i programmi che scrivono direttamente sui registri CRTC su MDA o su CGA probabilmente si blocchino su EGA.

Il CRTC dell'EGA supporta una più ampia gamma di funzioni di controllo di quello del 6845. Ad esempio, il CRTC può provocare un interrupt hardware all'inizio di un intervallo di soppressione verticale. Il CRTC supporta anche la visualizzazione simultanea di due aree non contigue del buffer del video (il Capitolo 12 descrive queste funzioni del CRTC).

Una funzione curiosa del CRTC dell'EGA è rappresentata dal registro di overflow (07H). Dal momento che l'EGA può visualizzare un reticolo di più di 256 linee, i registri del CRTC che contengono un numero di linee di scansione devono essere di 9 bit invece che di 8. Il bit più significativo di ognuno di questi registri è memorizzato nel registro di overflow.

Registro	Nome	Accesso Lettura/Scrittura EGA
00H	Totale orizzontale	Solo scrittura
01H	Fine abilitazione visualizzazione orizzontale	Solo scrittura
02H	Inizio soppressione orizzontale	Solo scrittura
03H	Fine soppressione orizzontale	Solo scrittura
04H	Inizio ritracciamento orizzontale	Solo scrittura
05H	Fine ritracciamento orizzontale	Solo scrittura
06H	Totale verticale	Solo scrittura
07H	Overflow	Solo scrittura
08H	Scansione riga preimpostata	Solo scrittura
09H	Indirizzo di linea di scansione massima	Solo scrittura
0AH	Inizio cursore	Solo scrittura
0BH	Fine cursore	Solo scrittura
0CH	Inizio indirizzo alto	Lettura/scrittura
0DH	Inizio indirizzo basso	Lettura/scrittura
0EH	Locazione alta cursore	Lettura/scrittura
0FH	Locazione bassa cursore	Lettura/scrittura
10H	Inizio ritracciamento verticale	Solo scrittura
10H	Penna ottica alta	Solo lettura
11H	Fine ritracciamento verticale	Solo scrittura
11H	Penna ottica bassa	Solo lettura
12H	Fine attivazione visualizzazione verticale	Solo scrittura
13H	Offset (larghezza linea logica)	Solo scrittura
14H	Locazione trattino sottolineatura	Solo scrittura
15H	Inizio soppressione verticale	Solo scrittura
16H	Fine soppressione verticale	Solo scrittura
17H	Controllo modalità	Solo scrittura
18H	Confronto linea	Solo scrittura

Figura 6. *Registri dati del controller del CRT dell'EGA e della VGA.*

MCGA

Nell'MCGA, le funzioni di un CRTC sono integrate in un componente circuitale denominato Memory Controller Gate Array. I primi 16 registri del controller della memoria sono analoghi a quelli del 6845 (vedere Figura 7). Come su CGA, tutti i registri del controller della memoria dell'MCGA compresi i registri del CRTC sono indexati tramite un registro di indirizzo alla porta di I/O 3D4H. Ai registri di dati stessi si accede alla porta 3D5H. Molte sono le funzioni che contraddistinguono il CRTC dell'MCGA dal 6845 del CGA. Tutti i registri del controller della memoria possono essere sia letti sia scritti. Inoltre, i registri da 00H a 07H possono essere indicati come di sola lettura in modo che i parametri di temporizzazione orizzontale e verticale non vengano inavvertitamente compromessi. Se si imposta a 1 il bit 7 del registro (10H) del controllo della modalità del controller della memoria si proteggono i registri da 00H a 07H.

Un'altra funzione del CRTC dell'MCGA è che l'hardware può calcolare i parametri di temporizzazione orizzontale per ogni modalità di visualizzazione disponibile. Quando il bit 3 del registro del controllo della modalità è impostato a 1, e quando i valori dei registri da 00H a 03H rappresentano valori di temporizzazione orizzontale appropriati per la modalità alfanumerica 40 per 25 (modalità 0 del BIOS del video), il controller della memoria genera segnali corretti di temporizzazione orizzontale in tutte le modalità di visualizzazione disponibili.

Se confrontate il CRTC dell'MCGA con il 6845 Motorola registro per registro, noterete diverse discrepanze nell'interpretazione dei valori memorizzati in alcuni registri del CRTC. In particolare, i valori previsti nei registri 09H, 0AH e 0BH sono specificati in unità di due linee di scansione sull'MCGA, invece che in unità di una linea di scansione come avviene con il 6845. Dal momento che la matrice di carattere alfanumerico di default dell'MCGA è alta 16 linee di scansione, questa funzione fornisce una certa compatibilità a basso livello, permettendovi di utilizzare per questi registri valori identici a quelli che utilizzereste su CGA.

Registro	**Nome**	**Accesso Lettura/Scrittura**
00H	Totale orizzontale	Lettura/scrittura
01H	Visualizzato orizzontalmente	Lettura/scrittura
02H	Inizio sincronizzazione orizzontale	Lettura/scrittura
03H	Larghezza impulso di sincronizzazione	Lettura/scrittura
04H	Totale verticale	Lettura/scrittura
05H	Regolazione totale verticale	Lettura/scrittura
06H	Visualizzato verticalmente	Lettura/scrittura
07H	Inizio sincronizzazione verticale	Lettura/scrittura
08H	(riservato)	
09H	Linee di scansione per carattere	Lettura/scrittura
0AH	Inizio cursore	Lettura/scrittura
0BH	Fine cursore	Lettura/scrittura
0CH	Inizio indirizzo alto	Lettura/scrittura
0DH	Inizio indirizzo basso	Lettura/scrittura
0EH	Locazione alta cursore	Lettura/scrittura
0FH	Locazione bassa cursore	Lettura/scrittura
10H	Controllo modalità	Lettura/scrittura
11H	Controllo interrupt	Lettura/scrittura
12H	Generatore carattere, polarità di sincronizzazione	Lettura/scrittura
13H	Puntatore generatore carattere	Lettura/scrittura
14H	Contatore generatore carattere	Lettura/scrittura
20-3FH	(riservati)	

Figura 7. *Registri dati del controller di memoria dell'MCGA. I registri da 00H a 0FH sono paragonabili a quelli del controller del CRT del CGA.*

VGA

Da un punto di vista funzionale, i registri del CRTC della VGA (vedere Figura 6) comprendono una serie ampliata dei registri del CRTC dell'EGA. La serie di registri del CRTC della VGA è indirizzabile alle stesse porte di I/O del CRTC dell'EGA. Alla serie di registri sono stati aggiunti alcuni campi di bit, con lo scopo primario di permettere al CRTC di gestire reticoli di 400 e di 480 linee. Tuttavia, a differenza del CRTC dell'EGA, il CRTC della VGA non supporta la penna ottica.

La cosa più importante, però, è che tutte le specifiche di registro del CRTC dell'EGA sono state riportate nella VGA. Di conseguenza, i programmi che scrivono sui registri del CRTC dell'EGA possono essere eseguiti, senza apportare modifiche, sull'hardware basato su VGA.

Come su MCGA, i registri dati del CRTC della VGA possono tutti essere letti e scritti. Inoltre, i registri di temporizzazione orizzontale e verticale (i registri da 00H a 07H del CRTC) possono essere protetti in scrittura impostando a 1 il bit 7 del registro di fine ritracciamento verticale (11H).

CONSIGLIO

Come nel caso degli adattatori Hercules, potete programmare il CRTC dell'EGA, dell'MCGA e della VGA tramite un'operazione di scrittura di porta a 16 bit (*OUT DX,AX*). Inoltre, scoprirete con la pratica che le operazioni di scrittura di porta a 16 bit funzionano su molti adattatori video non IBM. Non utilizzate però questa tecnica sugli MDA, sui CGA e sui cloni se la portabilità rappresenta per voi un fattore importante.

Calcoli di base del CRTC

Per utilizzare efficacemente il CRTC, dovete essere in grado di eseguire i calcoli di base necessari per specificare le temporizzazioni del CRTC in modo corretto. Questi calcoli sono basati su tre principi: la larghezza di banda del segnale video inviato al monitor e i valori di sincronizzazione orizzontale e verticale del monitor.

Clock del punto

I sottosistemi di visualizzazione del PC IBM visualizzano i pixel ad una velocità determinata dall'hardware. Questa velocità è conosciuta sotto diversi nomi: larghezza di banda del video, velocità di punto o velocità di pixel; l'oscillatore che genera questa velocità è conosciuto come clock del punto. L'MDA, il CGA e l'adattatore Hercules utilizzano solo un clock del punto; sull'EGA e sulla VGA sono disponibili più clock di punto

(vedere Figura 8). Maggiore è la frequenza del clock del punto, migliore sarà la risoluzione del pixel visualizzato.

Dopo aver stabilito una velocità di punto, il CRTC deve essere programmato in modo che le frequenze di scansione orizzontale e verticale inviate al video siano limitate alle frequenze gestibili dal video. I monitor più vecchi, come il video monocromatico dell'IBM, sono progettati per gestire solo una velocità di scansione orizzontale e una velocità di scansione verticale. I monitor più recenti, come il MultiSync della NEC, possono sincronizzarsi con una gamma di velocità di scansione orizzontali e/o verticali.

Sottosistema IBM	**Larghezza di banda (velocità di punto) in MHz**	**Velocità di scansione orizzontale in KHz**	**Velocità di scansione verticale in Hz**
MDA, HGC			
720x350 monocr.	16,257	18,43	50
CGA			
640x200 colore	14,318	15,75	60
EGA			
640x350 colore	16,257	21,85	60
640x200 colore	14,318	15,75	60
720x350 monocr.	16,257	18,43	50
InColor			
720x350 color	19,000	21,80	60
MCGA			
640x400 monocr./colore	25,175	31,50	70
640x480 monocr./colore	25,175	31,50	60
VGA			
640x400 monocr./colore	25,175	31,50	70
720x400 monocr./colore	28,322	31,50	70
640x480 monocr./colore	25,175	31,50	60
640x350 monocr./colore	25,175	31,50	70

Figura 8. *Temporizzazioni di base dei sottosistemi di visualizzazione IBM.*

Temporizzazione orizzontale

Considerate come calcolereste i tipici valori di registro di CRTC mostrati nella Figura 9 per un MDA con un monitor monocromatico IBM. La larghezza di banda di visualizzazione dell'MDA (la velocità di punto) è 16,257 MHz; cioè, 16.257.000 punti al secondo. La velocità di scansione orizzontale del monitor monocromatico è 18,432 KHz (18.432 linee al secondo). Dividendo la velocità di punto per la velocità di scansione orizzontale

si ottengono 882 punti per linea. Ogni carattere visualizzato dall'MDA è largo 9 punti, quindi il numero totale dei caratteri di ogni riga è 882:9, cioè 98.

Questo valore viene utilizzato per programmare il registro del totale orizzontale del CRTC. Per il CRTC dell'MDA, un 6845 Motorola, il valore memorizzato nel registro del totale orizzontale deve essere inferiore di 1 rispetto al totale calcolato, cioè 97 (61H).

Registro	Nome	Parametro	Descrizione
00H	Totale orizzontale	97 (61H)	(caratteri totali per linea di scansione) - 1
01H	Visualizzato orizzontalmente	80 (50H)	Caratteri visualizzati in ogni linea di scansione
02H	Posizione sincronizzazione orizzontale	82 (52H)	Posizione nella linea di scansione dove inizia il ritracciamento orizzontale
03H	Larghezza sincronizzazione orizzontale	15 (0FH)	Durata dell'intervallo di ritracciamento orizzontale (clock di carattere)
04H	Totale verticale	25 (19H)	Totale righe di caratteri in una cornice
05H	Regolazione totale verticale	2	Linee di scansione restanti in una cornice
06H	Visualizzato verticalmente	25 (19H)	Righe di caratteri visualizzate in ogni cornice
07H	Posizione sincronizzazione verticale	25 (19H)	Posizione nella cornice dove inizia il ritracciamento verticale
08H	Modalità interlace	2	Sempre impostato a 2
09H	Linea scansione massima	13 (0DH)	(altezza di un carattere nelle linee di scansione) - 1

Figura 9. *Tipici parametri CRTC per l'adattatore video monocromatico.*

In termini di temporizzazione del CRTC, il valore del totale orizzontale descrive la quantità di tempo, in "clock di carattere", richiesto per completare una scansione orizzontale. Durante questo intervallo, vengono visualizzati effettivamente 80 caratteri (questo è il valore utilizzato per il registro di "Visualizzato orizzontalmente"). Gli altri 18 clock di carattere vengono impiegati per la sovrascansione orizzontale e per il ritracciamento orizzontale.

La durata dell'intervallo di ritracciamento orizzontale varia dal 10 al 15 percento del valore del totale orizzontale. Il valore esatto dipende dal sottosistema di visualizzazione. Sull'MDA, l'intervallo di ritracciamento orizzontale è impostato a 15 clock di carattere memorizzando questo valore nel registro di larghezza sincronizzazione orizzontale del CRTC. Ciò lascia 3 clock di carattere di sovrascansione orizzontale. Il segnale di ritracciamento orizzontale è programmato in modo che inizi 2 clock di carattere dopo l'ultimo crattere visualizzato all'estrema destra memorizzando il valore 82 (52H) nel registro di posizione sincronizzazione orizzontale del CRTC. Di conseguenza, ci sono 2 clock di carattere di sovrascansione orizzontale destra e 1 clock di carattere di sovrascansione sinistra.

CONSIGLIO

La modifica del valore del registro di posizione sincronizzazione orizzontale cambia la dimensione delle aree di sovrascansione destra e sinistra e quindi la posizione orizzontale del reticolo visualizzato. Ad esempio, per spostare verso sinistra il reticolo visualizzato, aumentate la dimensione dell'intervallo di sovrascansione destra aumentando il valore nel registro di posizione sincronizzazione orizzontale del CRTC.

Temporizzazione verticale

Analoghe considerazioni si applicano alla programmazione del CRTC per generare temporizzazioni verticali appropriate. Il valore nominale di scansione orizzontale del monitor monocromatico dell'MDA è 18,432 KHz (18.432 linee al secondo) con un valore di scansione verticale di 50 Hz (50 cornici al secondo), quindi il numero di linee di una cornice è 18.432:50, o 368. Dal momento che ogni carattere visualizzato è alto 14 linee, 25 righe di caratteri danno come risultato 350 linee. Il CRTC dell'MDA utilizza sempre 16 linee per il ritracciamento verticale; questo significa che ci sono 368-(350+16) = 2 linee di sovrascansione verticale.

La programmazione del CRTC segue questi calcoli. L'altezza di ogni carattere visualizzato viene specificata dal valore del registro di linea di scansione massima del CRTC. Dal momento che i caratteri sono alti 14 linee di scansione, il valore di linea di scansione massima è 13 (0DH). Considerati insieme, i valori di totale verticale (25 righe di caratteri) e regolazione totale verticale (2 linee di scansione) indicano il numero totale di linee di scansione di una cornice. Il numero di righe di caratteri visualizzate (25) è indicato nel registro di "visualizzato verticalmente". La posizione nella cornice dove inizia il ritrac-

ciamento (25) viene specificata dal valore del registro di posizione sincronizzazione verticale.

I CRTC dell'MCGA, dell'EGA e della VGA sono molto più complessi del CRTC del 6845 Motorola dell'MDA e del CGA. Ciononostante, i registri che controllano le temporizzazioni orizzontale e verticale nei più recenti sottosistemi di visualizzazione sono simili per quanto riguarda la nomenclatura e la funzionalità ai registri del 6845. I calcoli per i CRTC dell'MCGA, dell'EGA e della VGA derivano dalla velocità del punto, dalla dimensione del carattere e dalle capacità orizzontale e verticale del monitor, esattamente come nel caso dell'MDA e della CGA.

Il registro di stato del CRT

Tutti i sottosistemi di visualizzazione IBM hanno un registro di stato del CRT di sola lettura. Questo registro si trova alla porta di I/O 3BAH su MDA e sugli adattatori Hercules e a 3DAH su CGA e MCGA; su EGA e VGA, questo registro si trova a 3BAH nelle configurazioni monocromatiche e a 3DAH nelle configurazioni a colori. Generalmente, due degli otto bit di questo registro rispecchiano lo stato corrente dei segnali di temporizzazione orizzontale e verticale generati dal CRTC. Questi bit di stato possono essere utilizzati per sincronizzare gli aggiornamenti del buffer del video con il ciclo di ripristino dello schermo per ridurre al minimo le interferenze con l'immagine visualizzata (il Capitolo 3 contiene degli esempi di questo tipo di programmazione).

Sfortunatamente, l'esatta interpretazione dei bit di stato del registro di stato del CRT varia a seconda dei diversi sottosistemi di visualizzazione IBM (vedere Figura 10). Di conseguenza si dovrebbero progettare dei programmi per determinare su quale hardware stanno operando (vedere Appendice C) prima di tentare di utilizzare le informazioni di stato di questo registro.

Il Listato 7 mostra come i bit di stato del registro di stato del CRT vengono utilizzati per sincronizzare il funzionamento del programma con il ciclo di ripristino del video. Questa subroutine può essere impiegata su CGA per sincronizzare l'intervallo di soppressione orizzontale. La subroutine usa il bit 3 del registro di stato del CRT, che indica quando il segnale di sincronizzazione verticale è attivo, per effettuare la sincronizzazione con l'inizio di un ciclo di ripristino. I loop a *L01* e a *L02* mostrano come realizzare questa operazione.

Successivamente i loop a *L03* e a *L04* si sincronizzano con il segnale di abilitazione visualizzazione, utilizzando il bit 0 del valore di stato del CRT. Quando il segnale di abilitazione visualizzazione viene disattivato, il loop a *L05* decrementa il valore di CX durante l'intervallo di soppressione orizzontale, cioè mentre il segnale di abilitazione visualizzazione è disattivato. Il numero di iterazioni contate in CX può quindi essere utilizzato come valore di superamento tempo per determinare quando è stata tracciata l'ultima linea orizzontale della cornice (vedere Capitolo 3).

```
                TITLE   'Listato 7'
                NAME    HRTimeout
                PAGE    55,132

;
; Nome:         HRTimeout
;
; Funzione:     Determina valore superamento tempo per intervallo soppressione
;               orizzontale
;
; Chiamante:    Microsoft C:
;
;                       int HRTimeout();


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC _HRTimeout
_HRTimeout      PROC    near

                push    bp          ; preambolo usuale C per stabilire
                mov     bp,sp       ;  cornice di stack

                mov     ax,40h
                mov     es,ax       ; ES := segmento dati del BIOS video

                mov     dx,es:[63h] ; DX := porta registro indirizzo del CRTC
                add     dl,6        ; DX := porta per registro di stato del CRTC

; sincronizza con inizio del ciclo di ripristino

L01:            in      al,dx       ; AL := stato del CRTC
                test    al,8        ; test bit 3
                jz      L01         ; loop mentre non avviene ritracciamento
                                    ;  verticale

L02:            in      al,dx
                test    al,8
                jnz     L02         ; loop durante ritracciamento verticale

; sincronizza con scansione orizzontale e temporizza l'intervallo di
;  soppressione orizzontale

                mov     cx,0FFFFh   ; CX := contatore di loop

                cli                 ; disabilita interrupt

L03:            in      al,dx
                test    al,1
                jnz     L03         ; loop mentre abilitazione visualizzazione è
                                    ;  disattivata

L04:            in      al,dx
                test    al,1
                jz      L04         ; loop mentre abilitazione visualizzazione
                                    ;  è attiva
```

```
L05:            in      al,dx
                test    al,1
                loopnz  L05         ; decrementa CX ed esegue loop mentre
                                    ;  abilitazione visualizzazione è disattiva

                sti                 ; riabilita interrupt

                mov     ax,cx       ; AX := contatore di loop
                neg     ax
                shl     ax,1        ; AX := valore di superamento tempo

                mov     sp,bp       ; elimina cornice di stack e ritorna a C
                pop     bp
                ret

_HRTimeout      ENDP

_TEXT           ENDS

                END
```

Listato 7. *Temporizzazione dell'intervallo di vuoto orizzontale su CGA.*

Modalità di visualizzazione

Nonostante i limiti di tempo imposti dal clock del punto e dalle velocità di scansione orizzontale e verticale dei monitor disponibili, tutti i sottosistemi di visualizzazione IBM ad eccezione dell'MDA possono essere programmati con una vasta gamma di diversi parametri CRTC. Ciò rende disponibili diverse modalità di visualizzazione. Ogni modalità di visualizzazione è caratterizzata dalla propria risoluzione (il numero di caratteri o pixel visualizzati orizzontalmente e verticalmente), dal numero di colori diversi che possono essere visualizzati contemporaneamente e dal formato dei dati visualizzabili nel buffer del video.

Risoluzione

La risoluzione orizzontale e verticale in una modalità di visualizzazione è una funzione della velocità del punto oltre che delle velocità di scansione verticale ed orizzontale del monitor. Il numero di pixel visualizzati in ogni cornice corrisponde alla velocità del punto divisa per la velocità di scansione verticale. L'effettiva risoluzione verticale ed orizzontale dipende anche dalla velocità di scansione orizzontale.

Colori

Il numero e la varietà dei colori che possono essere visualizzati in una modalità di visualizzazione dipendono dal progetto dei componenti di decodificazione dell'attributo e del generatore del segnale video del sottosistema di visualizzazione. Il decodificatore di attributo utilizza i dati memorizzati nel buffer del video per controllare i segnali di colore e di luminosità prodotti dal generatore del segnale video. Stabilire una particolare modalità di visualizzazione comporta sempre la programmazione del decodificatore di attributo del sottosistema di visualizzazione, oltre che l'aggiornamento dei relativi parametri del CRTC.

Organizzazione del buffer del video

Anche il formato dei dati nella RAM del video caratterizza una modalità di visualizzazione. In tutti i sottosistemi per PC e per PS/2, le modalità di visualizzazione possono essere classificate come modalità alfanumeriche o grafiche, a seconda del formato dei dati del buffer del video. Nelle modalità alfanumeriche, i dati del buffer del video vengono formattati come sequenze di coppie di codice ASCII e di byte di attributo; quindi il generatore di carattere alfanumerico traduce i codici ASCII in caratteri visualizzabili, mentre i byte di attributo specificano i colori utilizzati per visualizzarli (vedere Capitolo 3). Nelle modalità grafiche, il buffer del video è organizzato con sequenza di campi di bit; i bit di ogni campo stabiliscono il colore di un particolare pixel sullo schermo.

Controllo hardware della modalità di visualizzazione

Stabilire una modalità di visualizzazione su un sottosistema di visualizzazione per PC o per PS/2 IBM generalmente richiede una specifica programmazione del controllo di modalità oltre alla specifica dei parametri del CRTC. Ad esempio, il generatore di caratteri alfanumerici deve essere abilitato nelle modalità alfanumeriche e disabilitato nelle modalità grafiche. Inoltre, il clock di carattere interno del sottosistema, che determina il numero di pixel generati per ogni codice di carattere alfanumerico letto dal buffer del video, può operare a diverse velocità nelle varie modalità di visualizzazione. Queste ed altre funzioni interne sono controllate dal caricamento di valori appropriati, per ogni modalità di visualizzazione su uno o più specifici registri di controllo modalità.

Figura 10. *Assegnazione bit del registro di stato del CRTC*

	Registri	Bit 7	Bit 3	Bit 2	Bit 1	Bit 0
MDA	3BA		Drive Video			1=sinc. orizzon.
HGC, HGC+,InColor	3BA	0=sinc. vert.	Drive Video		1=Trigger penna ottica	1=sinc. orizzon.
CGA	3DA		1=sinc. vert.	1Switch penna ottica Chiuso	1=Trigger penna ottica	0=Abilita display
EGA	3BA o 3DA		1=sinc. vert.	1Switch penna ottica Chiuso	1=Trigger penna ottica	0=Abilita display
VGA	3BA o 3DA		1=sinc. vert.			0=Abilita display
MCGA	3DA		1=sinc. vert.*			0=Abilita display

*0=sinc. verticale in modalità 640x480 a 2 colori

MDA

Il registro di controllo modalità dell'MDA è un registro di sola scrittura collegato alla porta 3B8H (vedere Figura 11). Solo tre degli otto bit di questo registro sono significativi. Il bit 0 è impostato a 1 all'accensione e deve restare sempre impostato a 1. Il bit 3, quando è impostato a 1, abilita il ripristino del video; se si imposta a 0 questo bit si provoca la cancellazione dello schermo. Il bit 5 è il bit di attivazione del lampeggiamento; esso controlla il lampeggiamento dei caratteri. Sull'MDA la maggior parte dei programmi lasciano sempre impostato il bit 3. Il Capitolo 3 spiega come utilizzare il bit 5 (il bit di attivazione lampeggiamento).

Bit	Impostazioni
0	1 = adattatore abilitato (dovrebbe essere sempre = 1)
1	(non usato, dovrebbe essere sempre = 0)
2	(non usato, dovrebbe essere sempre = 0)
3	1 = video abilitato
	0 = video disabilitato (schermo vuoto)
4	(non usato, dovrebbe essere sempre = 0)
5	1 = attributo di lampeggiamento abilitato
	0 = attributo di lampeggiamento disabilitato
6	(non usato, dovrebbe essere sempre = 0)
7	(non usato, dovrebbe essere sempre = 0)

Figura 11. *Impostazioni dei bit per il registro di controllo modalità dell'MDA (3B8H).*

CGA e MCGA

Il registro di controllo modalità su CGA e su MCGA si trova a 3D8H (vedere Figura 12). I cinque bit meno significativi controllano le temporizzazioni interne appropriate per le modalità di visualizzazione selezionate, mentre il bit 5 è il bit di abilitazione lampeggiamento come su MDA. Le configurazioni utili di bit per il registro di controllo modalità del CGA sono elencate nella Figura 13. Questi valori corrispondono alle modalità di visualizzazione del BIOS disponibili su CGA.

I registri di controllo modalità su CGA e su MCGA presentano due differenze. Una consiste nel fatto che il registro di controllo modalità dell'MCGA può essere letto oltre che scritto; il registro del CGA è di sola scrittura. L'altra differenza riguarda la funzione del bit 2. Su CGA, l'impostazione a 1 del bit 2 disabilita la componente di burst di colore del segnale di output del video composito. Ciò può migliorare la qualità della visualizzazione se state utilizzando un monitor composito verde o ambra con un CGA. Su MCGA, che non supporta un monitor composito, la funzione del bit 2 del registro di controllo modalità è quella di selezionare una delle due sorgenti per il colore di primo piano nelle modalità grafiche a 2 colori.

Bit	Impostazioni
0	1 = modalità alfanumeriche a 80 caratteri
	0 = modalità alfanumeriche a 40 caratteri
1	1 = modalità grafica con larghezza 320
	0 = (tutte le altre modalità)
2	1 = burst di colore disabilitato (solo CGA)
	1 = colore primo piano dal registro 7 del DAC video (solo MCGA)
	0 = burst di colore abilitato (solo CGA)
	0 = colore primo piano dal registro del DAC video specificato nei bit da 0 a 3 del registro di tavolozza (3D9H) (solo MCGA)
3	1 = video abilitato
	0 = video disabilitato (schermo vuoto)
4	1 = modalità grafiche con larghezza 640
	0 = (tutte le altre modalità)
5	1 = attributo di lampeggiamento abilitato
	0 = attributo di lampeggiamento disabilitato
6	(non usato, dovrebbe essere sempre = 0)
7	(non usato, dovrebbe essere sempre = 0)

Figura 12. *Impostazioni dei bit per il registro di controllo modalità del CGA e dell'MCGA (3D8H)*

Numero modalità BIOS	Descrizione	Valore per il registro di controllo modalità
0	alfanumerica 40x25 (burst di colore disabilitato)	00101100b (2CH)
1	alfanumerica 40x25	00101000b (28H)
2	alfanumerica 80x25 (burst di colore disabilitato)	00101101b (2DH)
3	alfanumerica 80x25	00101001b (29H)
4	grafica 320x200	00101010b (2AH)
5	grafica 320x200 (burst di colore disabilitato)	00101110b (2EH)
6	grafica 640x200	00011100b (1CH)
7	alfanumerica 80x25 (solo MDA)	00101001b (29H)
11H	grafica 640x480 (solo MCGA)	00011000b (18H)

Figura 13. *Opzioni del registro di controllo modalità dell'MDA, del CGA e dell'MCGA.*

L'MCGA ha due registri addizionali per il controllo della modalità che non sono previsti sul CGA. Il registro di controllo modalità del controller della memoria dell'MCGA (10H) alla porta 3D4H/3D5H seleziona le modalità grafiche 640 x 480 a 2 colori e 320 x 200 a 256 colori (vedere Figura 14). Un registro di controllo modalità esteso è collegato

alla porta di I/O 3DDH. Questo registro viene utilizzato solo durante l'avviamento a freddo della macchina; esso non ha alcun uso pratico nei programmi applicativi.

Bit	Impostazioni
0	1 = seleziona modalità 320x200 a 256 colori
	0 = (tutte le altre modalità)
1	1 = seleziona modalità 640x480 a 2 colori
	0 = (tutte le altre modalità)
2	(riservato)
3	1 = parametri di temporizzazione orizzontale calcolati per la modalità di visualizzazione
	0 = parametri di temporizzazione orizzontale come specificato nei registri + da 00 a 03H
4	1 = abilita clock del punto (dovrebbe essere sempre a 1)
5	(riservato)
6	Contrario del bit 8 del registro di visualizzato verticalmente (06H)
7	1 = registri di protezione in scrittura da 00 a 07H
	0 = permette l'aggiornamento dei registri da 00 a 07H

Figura 14. *Impostazioni dei bit per il registro di controllo modalità del controller della memoria dell'MCGA.*

HGC

La scheda grafica Hercules (HGC) ha due registri di controllo il cui contenuto modifica la configurazione della modalità di visualizzazione. Il registro di controllo modalità a 3B8H è funzionalmente compatibile con il registro di controllo modalità dell'MDA, ma collega funzioni addizionali di configurazione di modalità ai bit 1 e 7 (vedere Figura 15). Il bit 1, quando è impostato a 1, stabilisce temporizzazioni interne per la modalità grafica 720 x 348. L'impostazione a 1 del bit 7 quando l'adattatore si trova in modalità grafica produce la visualizzazione della seconda metà del buffer del video di 64 KB dell'adattatore a B800:0000. Questi bit non hanno alcuna funzione, tuttavia, a meno che i bit appropriati del registro del commutatore di configurazione dell'adattatore non siano impostati correttamente.

Il registro del commutatore di configurazione (3BFH) determina la funzione del registro di controllo di modalità a 3B8H (vedere Figura 16). Quando il bit 0 del registro del commutatore di configurazione è impostato a 0, l'HGC non può essere posta in modalità grafica, quindi anche il bit 1 del registro di controllo modalità deve essere impostato a 0. Il bit 1 del registro del commutatore di configurazione controlla l'indirizzamento del buffer del video quando l'adattatore viene utilizzato in combinazione con un CGA o con un compatibile (vedere di seguito).

Bit	Impostazioni
0	(non usato)
1	1 = modalità grafica 720x348
	0 = modalità alfanumerica 80x25
2	(non usato, dovrebbe essere sempre = 0)
3	1 = video abilitato
	0 = video disabilitato (schermo vuoto)
4	(non usato, dovrebbe essere sempre = 0)
5	1 = attributo di lampeggiamento abilitato
	0 = attributo di lampeggiamento disabilitato
6	(non usato, dovrebbe essere sempre = 0)
7	1 = buffer modalità grafica visualizzato da B800:0000 (pagina video 1)
	0 = buffer modalità grafica visualizzato da B000:0000 (pagina video 2)

Figura 15. *Impostazioni dei bit per il registro di controllo modalità Hercules (3B8H). Questo registro è identico a quello dell'HGC, dell'HGC+ e della scheda InColor.*

Bit	Impostazioni
0	1 = permette modalità grafica
	0 = impedisce modalità grafica
1	1 = abilita 32 KB superiori del buffer del video della modalità grafica a B800:0000
	0 = disabilita 32 KB superiori del buffer della modalità grafica
2-7	(non usati)

Figura 16. *Impostazioni dei bit per il registro di commutazione di configurazione Hercules (3BFH). Questo registro è identico a quello dell'HGC, dell'HGC+ e della scheda InColor.*

HGC+ e scheda InColor

L'HGC+ e la scheda InColor prevedono un registro di controllo modalità estesa (chiamato registro xMode) oltre ai registri di controllo modalità e di commutatore di configurazione dell'HGC. Il registro xMode è un registro di sola scrittura indirizzabile come registro 14H alla porta 3B4H/3B5H (il registro viene indirizzato esattamente come se fosse un registro CRTC). Il registro xMode controlla il generatore di caratteri alfanumerici; il Capitolo 10 spiega tutto questo in dettaglio.

EGA e VGA

Quando stabilite una modalità di visualizzazione su EGA e su VGA, potete controllare la temporizzazione interna e l'indirizzamento di diversi componenti del sottosistema di visualizzazione. Questi comprendono il sequencer, il controller grafico e il controller

di attributo, ognuno dei quali ha diversi registri di controllo. Esiste anche un registro di output vario, che controlla l'indirizzamento della porta di I/O e del buffer del video e seleziona la frequenza del clock del punto.

CONSIGLIO

Tutti i registri del sequencer, del controller grafico e del controller di attributo dell'EGA sono registri di sola scrittura, ma su VGA essi possono essere letti oltre che scritti.

Sequencer

Il sequencer genera le temporizzazioni interne per l'indirizzamento della RAM del video. Esso ha cinque registri di dati programmabili (vedere Figura 17) collegati alle porte 3C4H e 3C5H in un modo analogo al collegamento del registro del CRTC. Il registro di indirizzo del sequencer si trova a 3C4H; i suoi cinque registri di dati vengono selezionati memorizzando un valore indice tra 0 e 4 nel registro di indirizzo e quindi accedendo al corrispondente registro dati a 3C5H.

Registro	Nome
0	Rispristino
1	Modalità clock
2	Maschera di mappa
3	Selezione mappa carattere
4	Modalità memoria

Figura 17. *Registri del sequencer dell'EGA e della VGA.*

Controller grafico

Il controller grafico agisce da intermediario sul flusso dei dati tra il buffer del video e la CPU, oltre che tra il buffer del video e il controller di attributo. Il controller grafico ha nove registri di dati, oltre ad un registro di indirizzo (vedere Figura 18). Il registro di indirizzo è collegato alla porta 3CEH e i registri dei dati sono collegati alla porta 3CFH.

Registro	Nome
0	Impostazione/ripristino
1	Abilitazione impostazione/ripristino
2	Confronto colore
3	Rotazione dati/selezione funzione
4	Selezione lettura mappa

(continua)

Registro	Nome
5	Modalità grafica
6	Varie
7	Colore ininfluente
8	Maschera bit

Figura 18. *Registri del controller grafico dell'EGA e della VGA.*

Controller di attributo

Il controller di attributo supporta una tavolozza di 16 colori su EGA e VGA. Esso controlla anche il colore visualizzato durante gli intervalli di sovrascansione. Il registro di indirizzo del controller di attributo e 21 registri dati sono tutti collegati alla porta di I/O 3C0H (vedere Figura 19). Un valore scritto sulla porta 3C0H verrà memorizzato o nel registro di indirizzo o in un registro dati, a seconda dello stato del flip-flop interno del controller di attributo.

Registro(i)	Funzione
0-0FH	Tavolozza
10H	Controllo di modalità attributo
11H	Colore di sovrascansione
12H	Abilitazione piano colore
13H	Pianificazione pixel orizzontale
14H	Selezione colore (solo VGA)

Figura 19. *Registri del controller di attributo dell'EGA e della VGA.*

Per imostare il flip-flop, eseguite un'operazione di lettura di I/O (*IN AL,DX*) del registro di stato del CRT (porta 3BAH nelle modalità monocromatiche, 3DAH nelle modalità a colori). Il Listato 8 illustra come ciò viene realizzato aggiornando un registro del controller di attributo. Su VGA, i registri dei dati del controller di attributo possono essere letti oltre che scritti. Eseguite questa operazione scrivendo il numero di registro sulla porta 3C0H e quindi leggendo il valore dalla porta 3C1H.

```
; programma direttamente il controller di attributo

            mov     ax,40h
            mov     es,ax        ; ES := segmento dati BIOS di
                                 ;  visualizzazione
            mov     dx,es:[63h]  ; DX := 3x4h (3B4h o 3D4h)
            add     dl,6         ; DX := 3xAh (registro di stato del CRT)
```

```
            cli                 ; azzera gli interrupt
            in      al,dx       ; ripristina flip-flop controller di
                                ;  attributo
            push    dx          ; conserva porta del registro di stato
            mov     dl,0C0h     ; DX := 3C0h
            mov     al,RegNumber
            out     dx,al       ; scrive su registro di indirizzo
            jmp     $+2         ; esegue alcuni cicli in modo che il
                                ;  controller di attributo possa rispondere
            mov     al,DataValue
            out     dx,al       ; scrive su registro dati

            pop     dx          ; DX := 3xAh
            in      al,dx       ; ripristina flip-flop
            mov     dl,0C0h
            mov     al,20h      ; ripristina tavolozza
            out     dx,al
            sti                 ; abilita interrupt

; uso del BIOS di visualizzazione

            mov     ax,1000h    ; AH := 10h (numero funzione INT 10h)
                                ; AL := 0 (imposta registro individuale
                                ;  del controller di attributo)
            mov     bl,RegNumber
            mov     bh,DataValue
            int     10h
```

Listato 8. *Aggiornamento del registro del controller di attributo dell'EGA o della VGA.*

CONSIGLIO

Potete effettuare delle operazioni di scrittura di porta a 16 bit (*OUT DX,AX*) per memorizzare i dati nei registri del sequencer e del controller grafico dell'EGA e della VGA. Su EGA, potete utilizzare la stessa tecnica per programmare il controller di attributo, che riconosce le operazioni di scrittura di porta di I/O a 3C1H oltre che a 3C0H. Tuttavia, in questo caso il controller di attributo della VGA non emula l'EGA, di conseguenza questa tecnica dovrebbe essere impiegata con cautela quando la compatibilità con la VGA rappresenta un fattore importante.

Supporto BIOS del video

Il BIOS di visualizzazione supporta diverse modalità sui sottosistemi di visualizzazione dei PC e dei PS/2 IBM (vedere Figura 20). Le routine del BIOS di visualizzazione, che possono essere chiamate tramite INT 10H, permettono di stabilire una modalità di visualizzazione semplicemente specificando il numero relativo.

Non tutte le modalità di visualizzazione del BIOS sono disponibili su tutti i sottosistemi di visualizzazione dei PC IBM. Inoltre, il BIOS di visualizzazione non supporta le

configurazioni di modalità di visualizzazione su hardware non IBM a meno che questo non emuli esattamente l'hardware IBM corrispondente.

Ad esempio, tutti gli adattatori video Hercules emulano esattamente l'MDA dell'IBM. Di conseguenza, il BIOS di visualizzazione può essere utilizzato per selezionare la modalità alfanumerica monocromatica (modalità 7 del BIOS) su tutti i prodotti Hercules. L'hardware Hercules però supporta anche una modalità grafica 720 per 348 che non viene riconosciuta dal BIOS di visualizzazione dell'IBM. Di conseguenza, per impostare la modalità grafica Hercules, un programma dovrà configurare direttamente l'hardware (vedere il Listato 9).

Numero esad. mod.	Risoluzione	Colori	Tipo modalità	Segmento buffer	MDA	CGA	EGA	MCGA	VGA
0	40x25 car. (320x200 pixel)+*	16	Alfanum.	B800		x	x	x	x
0	40x25 car. (320x350 pixel)+	16	Alfanum.	B800			x		x
0	40x25 car. (320x400 pixel)	16	Alfanum.	B800				x	
0	40x25 car. (360x400 pixel)+	16	Alfanum.	B800					x
1	40x25 car. (320x200 pixel)+	16	Alfanum.	B800		x	x	x	x
1	40x25 car. (320x350 pixel)+	16	Alfanum.	B800			x		x
1	40x25 car. (320x400 pixel)	16	Alfanum.	B800				x	
1	40x25 car. (360x400 pixel)+	16	Alfanum.	B800					x
2	80x25 car. (640x200 pixel)+	16	Alfanum.	B800		x	x	x	x
2	80x25 car. (640x350 pixel)+	16	Alfanum.	B800			x		x
2	80x25 car. (640x400 pixel)	16	Alfanum.	B800				x	
2	80x25 car. (720x400 pixel)+	16	Alfanum.	B800					x
3	80x25 car. (640x200 pixel)+	16	Alfanum.	B800		x	x	x	x
3	80x25 car. (640x350 pixel)+	16	Alfanum.	B800			x		x
3	80x25 car. (640x400 pixel)	16	Alfanum.	B800				x	
3	80x25 car. (720x400 pixel)+	16	Alfanum.	B800					x
4	320x200 pixel	4	Grafica	B800		x	x	x	x
5	320x200 pixel++	4	Grafica	B800		x	x	x	x
6	640x200 pixel	2	Grafica	B800		x	x	x	x
7	80x25 car. (720x350 pixel)+	2	Alfanum.	B000	x		x		x
7	80x25 car. (720x400 pixel)+	2	Alfanum.	B000					x

(continua)

Numero esad. mod.	Risoluzione	Colori	Tipo modalità	Segmento buffer	MDA	CGA	EGA	MCGA	VGA
8	(solo PCjr)								
9	(solo PCjr)								
0A	(solo PCjr)								
0B	(usato da BIOS del video EGA)								
0C	(usato da BIOS del video EGA)								
0D	320x200 pixel	16	Grafica	A000			x		x
0E	640x200 pixel	16	Grafica	A000			x		x
OF	640x350 pixel	2	Grafica	A000			x		x
10	640x350 pixel**	4	Grafica	A000			x		
10	640x350 pixel	16	Grafica	A000			x		x
11	640x480 pixel	2	Grafica	A000				x	x
12	640x480 pixel	16	Grafica	A000					x
13	320x200 pixel	256	Grafica	A000				x	x

* Su CGA, la componente di burst di colore del segnale di video composito è disabilitata. Ciò migliora l'aspetto della visualizzazione monocromatica verde o ambra. Su EGA, MCGA e VGA la modalità 0 è identica alla modalità 1, e la modalità 2 è identica alla modalità 3.

+ Su VGA, la risoluzione verticale dei pixel in questa modalità viene selezionata utilizzando la funzione INT 10H (vedere Appendice A).

++ Su CGA, il burst di colore è disabilitato e la tavolozza a quattro colori contiene nero, ciano, rosso e bianco (per dettagli, vedere il Capitolo 4). Su EGA, MCGA e VGA, la modalità 5 è identica alla modalità 4.

** Su EGA con solo 64 KB di RAM video è possibile visualizzare solamente quattro colori.

Figura 20. *Modalità di visualizzazione del BIOS ROM.*

```
                TITLE   'Listato 9'
                NAME    HercGraphMode
                PAGE    55,132

;
; Nome:         HercGraphMode
;
; Funzione:     Impostare modalità grafica Hercules 720x348 su HGC, HGC+,
;               InColor
;
; Chiamante:    Microsoft C:
;
;                                   annulla HercGraphMode();
;

DGROUP          GROUP   _DATA

_TEXT           SEGMENT  byte public 'CODE'
                ASSUME   cs:_TEXT,ds:DGROUP

                PUBLIC  _HercGraphMode
_HercGraphMode PROC     near

                push     bp        ; mantiene registri chiamante
                mov      bp,sp
                push     si
                push     di

; Aggiorna area dati BIOS di visualizzazione con valori corretti

                mov      ax,40h
                mov      es,ax
                mov      di,49h    ; ES:DI := 0040:0049 (area dati BIOS)

                mov      si,offset DGROUP:BIOSData
                mov      cx,BIOSDataLen
                rep      movsb     ; aggiorna area dati BIOS

; Imposta switch configurazione

                mov      dx,3BFh   ; DX := porta dello switch di configurazione
                mov      al,1      ; AL bit 1 := 0 (esclude secondi 32K del
                                   ;               buffer del video)
                                   ; AL bit 0 := 1 (permette impostazione
                out      dx,al     ;               modalità grafica tramite
                                   ;               3B8h)

; Cancella schermo per evitare interferenze durante programmazione CRTC

                mov      dx,3B8h   ; DX := porta reg. di contr. modal. CRTC
                xor      al,al     ; AL bit 3 := 0 (disabilita segnale video)
                out      dx,al     ; cancella lo schermo

; Programma il CRTC

                sub      dl,4      ; DX := porta 3B4h reg. indirizzo CRTC
```

```
                mov     si,offset DGROUP:CRTCParms
                mov     cx,CRTCParmsLen

L01:            lodsw                   ; AL := numero di registro del CRTC
                                        ; AH := dati per questo registro
                out     dx,ax
                loop    L01

; Imposta modalità grafica

                add     dl,4            ; DX := 3B8h (reg. controllo modal. CRTC)
                mov     al,CRTMode        ; AL bit 1 = 1 (abilita modalità grafica)
                                          ; bit 3 = 1 (abilita video)
                out     dx,al

                pop     di              ; ripristina registri ed esce
                pop     si
                mov     sp,bp
                pop     bp
                ret

_HercGraphMode ENDP

_TEXT           ENDS


_DATA           SEGMENT word public 'DATA'

                                        ; Questi sono i parametri consigliati da
                                        ;  Hercules.
                                        ; Sono basati su 16 pixel/carattere e
                                        ; 4 linee di scansione per carattere.

CRTCParms       DB      00h,35h         ; Totale orizzontale :  54 caratteri
                DB      01h,2Dh         ; Visualizz. orizzontale :  45 caratteri
                DB      02h,2Eh         ; Posizione sincron. orizz.: al 46esimo
                                        ;  carattere
                DB      03h,07h         ; Ampiezza sincroniz. oriz. :  7 clock di
                                        ;  carattere

                DB      04h,5Bh         ; Totale vertic. :  92 caratteri (368 linee)
                DB      05h,02h         ; Regolaz. vert. :  2 linee di scansione
                DB      06h,57h         ; Visual. vert. :  87 righe di caratteri
                                        ;  (348 linee)
                DB      07h,57h         ; Posiz. sincr. vertic. :  dopo 87esima
                                        ;  riga di carattere

                DB      09h,03h         ; Linea scans. max :  4 linee scansione per
                                        ;  carattere
CRTCParmsLen EQU        ($-CRTCParms)/2

BIOSData        DB      7               ; CRT_MODE
                DW      80              ; CRT_COLS
                DW      8000h           ; CRT_LEN
                DW      0               ; CRT_START
```

```
                DW      8 dup(0) ; CURSOR_POSN
                DW      0        ; CURSOR_MODE
                DB      0        ; ACTIVE_PAGE
CRTCAddr        DW      3B4h     ; ADDR_6845
CRTMode         DB      0Ah      ; CRT_MODE_SET (valore per porta 3B8h)
                DB      0        ; CRT_PALETTE (non usato)

BIOSDataLen     EQU     $-BIOSData

_DATA           ENDS

                END
```

Listato 9. *Configurazione di un adattatore Hercules per la modalità grafica 720 per 348.*

Combinazioni di sottosistemi di visualizzazione

L'IBM ha progettato l'MDA e il CGA originali in modo tale che entrambi gli adattatori possano essere impiegati sullo stesso PC. Questo è possibile in quanto i registri e altri controlli del CRTC e i registri di stato sono assegnati ad una gamma di porte di I/O su MDA diversa da quella su CGA. Gli indirizzi di porta dell'MDA variano da 3B0H a 3BFH, mentre quelli del CGA variano da 3D0H a 3DFH. Inoltre, i buffer del video su MDA e su CGA occupano aree diverse dello spazio di indirizzamento dell'80x86: il buffer del video da 4 KB dell'MDA si trova a B000:0000, mentre il buffer da 16 KB del CGA inizia a B800:0000.

Questa separazione fu riportata nel progetto dell'EGA. L'indirizzamento del buffer del video e della porta di I/O dell'EGA sono programmabili. Quando l'EGA è collegato ad un monitor monocromatico, vengono utilizzati gli indirizzi compatibili MDA. Quando l'EGA viene utilizzato con un monitor a colori, vengono utilizzati gli indirizzi compatibili CGA. Di conseguenza, un EGA può coesistere sia con un MDA sia con un CGA.

La Figura 21 mostra quale sottosistema di visualizzazione di PC o di PS/2 può coesistere nello stesso computer. La tabella rispecchia la dicotomia tra l'indirizzamento di buffer del video e di porta di I/O dei compatibili MDA e dei compatibili CGA. Come regola generale, di solito si può combinare un adattatore compatibile MDA con un adattatore compatibile CGA nello stesso sistema.

NOTA: La scheda InColor Hercules dovrebbe essere considerata un adattatore compatibile MDA, anche se è palesemente una scheda a colori. In effetti, se utilizzate la scheda InColor su un modello 30 PS/2 con un monitor monocromatico collegato all'MCGA del modello 30, finirete per avere la strana combinazione di un sottosistema a colori compatibile MDA e di un sottosistema monocromatico compatibile CGA all'interno dello stesso computer.

Le routine delle modalità di visualizzazione del BIOS generalmente supportano configurazioni a doppio monitor. Le routine BIOS di visualizzazione utilizzano i bit 4 e 5

della variabile EQUIP_FLAG a 0040:0010 nell'area dati di visualizzazione del BIOS per scegliere tra due sottosistemi di visualizzazione. Se si verificano dei conflitti di indirizzamento tra i due sottosistemi, il BIOS dell'MCGA e della VGA fornisce un'interfaccia di "commutatore di visualizzazione" che permette di disabilitare ed abilitare ogni sottosistema individualmente (vedere Appendice A).

	MDA	CGA	EGA	MCGA	Adattatore VGA	HGC	HGC+	InColor
MDA		x	x	x	x			
CGA	x		x			x	x	x
EGA	x	x				x	x	x
MCGA	x				x	x	x	x
Adatt. VGA	x			x		x	x	x
HGC		x	x	x	x			
HGC+		x	x	x	x			
InColor		x	x	x	x			

Figura 21. *Combinazioni possibili dei sottosistemi di visualizzazione per PC e PS/2 IBM.*

Con alcune combinazioni di adattatori video, lo spazio di indirizzo occupato dai buffer del video dei due sottosistemi di visualizzazione potrebbe sovrapporsi anche se le loro assegnazioni di indirizzo di porta di I/O non si sovrappongono. In una simile situazione dovrete escludere selettivamente una parte o tutto il buffer del video di un sottosistema dalla mappa di memoria della CPU in modo che la CPU possa accedere al buffer dell'altro sottosistema senza conflitti di indirizzamento. La tecnica per realizzare questa esclusione varia a seconda dell'hardware.

MDA

Il buffer del video dell'MDA è collegato agli indirizzi tra B000:0000 e B000:FFFF. Lo stesso buffer è anche collegato ai blocchi di 4 KB di RAM che iniziano ai segmenti B100H, B200H e così via fino a B700H, sebbene non ci sia motivo per cui il software debba utilizzare queste mappe di indirizzo alternative. La mappatura di indirizzo del buffer del video dell'MDA non può essere disabilitata.

Hercules

Su HGC, HGC+ e sulla scheda InColor, il buffer del video occupa i 64 KB di RAM che iniziano a B000:0000. I secondi 32 KB del buffer del video si sovrappongono allo spazio di indirizzo del buffer del video del CGA (a partire da B800:0000). Per questo motivo questi adattatori Hercules sono progettati in modo che i secondi 32 KB possano essere esclusi selettivamente dalla mappa di memoria della CPU. L'estensione dello spazio di indirizzo del buffer del video dipende dal valore da voi memorizzato nel registro di switch di configurazione (3BFH). Quando il bit 1 di questo registro è impostato a 0 (il default all'accensione), la RAM del video occupa gli indirizzi da B000:0000 a B000:7FFF, il che esclude l'area dei secondi 32 KB dalla mappa di memoria della CPU e permette alla scheda di essere utilizzata con un CGA. Per rendere indirizzabile la seconda metà del buffer del video, impostate il bit 1 a 1.

CGA

Il buffer del video del CGA si collega agli indirizzi tra B800:0000 e B800:3FFF. Lo stesso buffer è anche collegato tra BC00:0000 e BC00:3FFF, sebbene pochi programmi utilizzino questa mappa di indirizzo alternativa. Come nel caso dell'MDA, il collegamento del buffer del video del CGA non può essere modificato.

Questo non è il caso, però, di tutti i cloni CGA. La scheda a colori Hercules (da non confondersi con la scheda InColor) è una scheda analoga al CGA, il cui buffer può essere escluso dallo spazio di indirizzo della CPU. Questo viene realizzato impostando a 1 il bit 1 del registro dello switch di configurazione (3BFH) della scheda. Questo registro si collega alla stessa porta di I/O esattamente come il registro equivalente dell'HGC, dell'HGC+ o della scheda InColor, ma la polarità del bit di controllo è invertita rispetto a quella delle altre schede Hercules. Di conseguenza, commutando questo bit, il software può indirizzare i buffer di video sia sulla scheda a colori Hercules sia su un altro adattatore Hercules senza conflitti di indirizzamento.

EGA

Il buffer del video dell'EGA può essere collegato ad una qualsiasi delle quattro locazioni, a seconda dei valori dei bit 2 e 3 del registro "varie" del controller grafico (vedere la Figura 22). I valori di default di questi bit dipendono dalla modalità video. Quando il BIOS del video imposta una modalità di visualizzazione, esso imposta questi bit sui valori appropriati.

L'EGA fornisce anche un altro livello di controllo sulla mappa di indirizzo del buffer del video. Quando viene impostato a 0, il bit 1 del registro di output varie dell'EGA (3C2H) esclude l'intero buffer del video dallo spazio di indirizzo della memoria della CPU.

Bit 3	Bit 2	Gamma di indirizzamento del buffer del video
0	0	A000:0000-B000:FFFF
0	1	A000:0000-A000:FFFF
1	0	B000:0000-B000:7FFF
1	1	B800:0000-B800:7FFF

Figura 22. *Controllo dell'indirizzamento del buffer del video di EGA e VGA con il registro varie del controller grafico.*

MCGA

Il buffer del video di 64 KB dell'MCGA occupa gli indirizzi tra A000:0000 e A000:FFFF, ma i secondi 32 KB del buffer, a partire da A000:8000 (A800:0000), si collegano anche alla gamma di indirizzo del buffer del video del CGA (da B800:0000 a B800:7FFF). L'indirizzamento della CPU del buffer del video e delle porte di I/O dell'MCGA può essere disabilitato impostando a 0 il bit 2 della porta di controllo della piastra di sistema a 65H. Il Listato 10 mostra come la funzione 12H di INT 10H possa essere chiamata per impostare o azzerare questo bit.

```
mov     ah,12h      ; AH := 12h (numero funzione INT 10h)
mov     al,1        ; AL := 1 (disabilita indirizzamento)
                    ; (usa AL = 0 per abilitare indirizzamento)
mov     bl,32h      ; numero sottofunzione INT 10H
int     10h

cmp     al,12h
jne     ErrorExit   ; salta se il BIOS non supporta questa
                    ;  funzione
```

Listato 10. *Abilitazione e disabilitazione dell'indirizzamento della porta di I/O e del buffer del video su MCGA o VGA.*

VGA

Il controllo sulla mappa di indirizzo del buffer del video della VGA è identico a quello dell'EGA. Tuttavia, esistono due diversi metodi per disabilitare l'indirizzamento della CPU del sottosistema di visualizzazione, a seconda che si utilizzi una VGA integrata (nei modelli 50, 60 o 80 del PS/2) o l'adattatore VGA. Nel sottosistema integrato, il registro di abilitazione sottosistema di visualizzazione (3C3H) controlla sia l'indirizzamento del buffer del video sia l'indirizzamento della porta di I/O; l'impostazione a 0 del bit 0 di questo registro disabilita l'indirizzamento, e l'impostazione a 1 del bit 0 lo abilita.

Sull'adattatore VGA non esiste il registro di abilitazione sottosistema di visualizzazione, ma è il bit 3 del registro di controllo alla porta di I/O 46E8H che abilita e disabilita

l'indirizzamento: la scrittura di un valore di default di 0EH su questa porta abilita l'indirizzamento; la scrittura di un valore di 6 lo disabilita.

In tutti i sottosistemi VGA, però, la funzione 12H di INT 10H fornisce la stessa interfaccia fornita su MCGA (vedere Listato 10). A causa delle differenze hardware tra l'MCGA, la VGA integrata e l'adattatore VGA, è più semplice utilizzare la funzione 12H di INT 10H per abilitare o disabilitare l'indirizzamento nei sottosistemi di visualizzazione dei PS/2 (vedere Listato 10).

3
Modalità alfanumeriche

Uso delle modalità alfanumeriche
Supporto del BIOS e del sistema operativo
Velocità - Compatibilità

Rappresentazione dei dati alfanumerici

Attributi
MDA - HGC - CGA - EGA
Scheda InColor - MCGA - VGA

Calcolo della scala dei grigi

Colore del bordo
CGA - EGA e VGA

Come evitare l'effetto neve del CGA
Cancellazione dello schermo
Uso dell'intervallo di soppressione verticale
Uso dell'intervallo di soppressione orizzontale

Uso dell'intero buffer del video
Pagine di visualizzazione del CGA
Pagine video dell'EGA, dell'MCGA e della VGA

Controllo del cursore
Dimensione del cursore su MDA e CGA
Locazione del cursore su MDA e CGA
Controllo del cursore su MCGA - Controllo del cursore su EGA e VGA
Emulazione del cursore del BIOS ROM
Un cursore invisibile

Tutti i sottosistemi di visualizzazione dei PC e dei PS/2 IBM ad eccezione dell'MDA possono essere programmati per visualizzare caratteri sia nelle modalità alfanumeriche sia nelle modalità grafiche. Questo capitolo esamina quanto è necessario conoscere per utilizzare le modalità alfanumeriche (i vantaggi e gli inconvenienti della programmazione nelle modalità alfanumeriche, i principi fondamentali dei colori, del lampeggiamento e di altri attributi di visualizzazione carattere oltre a speciali tecniche che sfruttano le capacità dell'hardware per migliorare l'aspetto su schermo e la prestazione dei vostri programmi).

Uso delle modalità alfanumeriche

Il BIOS di visualizzazione di tutti i PC e i PS/2 IBM seleziona sempre una modalità alfanumerica di visualizzazione all'accensione del computer. Nella famiglia di PC IBM, degli switch posti sulla piastra madre, sull'adattatore video o su entrambi, determinano se è stata selezionata una modalità a 40 o a 80 colonne e se viene impiegato un monitor a colori o monocromatico. Nella serie PS/2, la modàlità di visualizzazione iniziale è sempre una modalità alfanumerica a 80 colonne. Inoltre, la modalità di visualizzazione impostata dal BIOS ROM è quella utilizzata inizialmente dal sistema operativo. Fino a che non eseguite un programma che modifica la modalità di visualizzazione, tutto l'output del video apparirà nella modalità di default (alfanumerica).

Per questo motivo, il metodo più semplice per scrivere un programma è quello di assumere che verrà eseguito in una modalità alfanumerica e di programmare di conseguenza l'interfaccia video. Questo presupposto riduce al minimo i codici richiesti per inviare l'output allo schermo. Non solo le routine di output su video alfanumeriche risulteranno più semplici di routine equivalenti per modalità grafiche, ma nella maggior parte dei casi il BIOS ROM o il sistema operativo fornisce routine di output carattere che possono essere utilizzate in una qualsiasi modalità alfanumerica.

Supporto del BIOS e del sistema operativo

Nel PC IBM, le routine di output su video del sistema operativo sono di solito basate su una serie di semplici routine del BIOS ROM chiamate dall'interrupt software 10H. Potete inviare caratteri allo schermo sia utilizzando le chiamate di sistema operativo sia chiamando direttamente le routine di INT 10H. In entrambi i casi, l'uso di queste routine evita la necessità di scrivere proprie routine di output caratteri.

Un ulteriore vantaggio nell'uso delle funzioni di output carattere del sistema operativo o del BIOS risiede nel fatto che i programmi che impiegano solo queste funzioni sono quelli che più probabilmente funzioneranno su diversi tipi di hardware di visualizzazione. Ad esempio, un programma che utilizza solo le chiamate di funzione dell'MS-DOS per l'output su video funzionerà in quasi tutti gli ambienti operativi MS-DOS, a prescin-

dere dall'hardware di visualizzazione utilizzato, compresa (ma senza limitarsi ad essa) l'intera famiglia dei PC e dei PS/2 IBM.

Naturalmente, l'indirizzamento dell'output allo schermo tramite sistema operativo è relativamente lento in confronto alla scrittura diretta sull'hardware. L'impiego delle routine di output carattere del sistema operativo introduce una certa quantità di operazioni ausiliarie inevitabili, in particolare quando sono supportate funzioni come la ridirezione dell'input/output e la multielaborazione. Ciononostante, queste operazioni di servizio possono essere considerate accettabili in molte applicazioni. Dovrete sempre valutare se la programmazione supplementare e la diminuzione di trasportabilità richieste per migliorare le prestazioni di output su video siano più importanti relativamente alla vostra applicazione.

Velocità

Ciò non significa che l'output alfanumerico su schermo sia intrinsecamente lento. Se viene confrontato all'output di caratteri nelle modalità grafiche, l'output alfanumerico risulta significativamente più rapido, semplicemente perché per visualizzare i caratteri devono essere memorizzati molto meno dati nel buffer del video. Nelle modalità alfanumeriche, ogni carattere viene rappresentato da una singola parola a sedici bit; l'hardware di visualizzazione si occupa di visualizzare i pixel che costituiscono il carattere. Nelle modalità grafiche, ogni pixel di ogni carattere viene rappresentato esplicitamente da un campo di bit nel buffer del video. Per questo motivo, l'output in modalità grafica è molto più dispendioso dell'output di carattere equivalente nelle modalità alfanumeriche, sia in termini di memoria di visualizzazione utilizzata sia in termini di elaborazione richiesta.

Ad esempio, in una modalità grafica a 16 colori, ogni carattere tracciato sullo schermo in una matrice di punti da 8 per 8 viene rappresentato da 32 byte di dati nel buffer del video (8x8x4 bit per pixel). Le operazioni ausiliarie della memoria aumentano rapidamente, in stretta relazione all'aumentata risoluzione e all'aggiunta di più colori, come pure aumenta la quantità di tempo impiegato dalla CPU per manipolare i dati nel buffer del video. Sui più recenti adattatori video, coprocessori grafici dedicati come l'82786 o il TI 34010 dell'~~Intel~~* possono assumersi la gran parte del carico di calcoli necessari per la visualizzazione del testo in modalità grafica, aumentando così la velocità dell'output del testo in questa modalità. Senza coprocessore, però, l'output nelle modalità grafiche risulta molto più lento che nelle modalità alfanumeriche.

Compatibilità

La scrittura di un programma che è compatibile con diversi sottosistemi di visualizzazione IBM è più semplice se utilizzate solo modalità di visualizzazione alfanumeriche. La ragione è semplice: tutti i sottosistemi di visualizzazione IBM comunemente utilizzati supportano la modalità alfanumerica 80 colonne per 25 righe con lo stesso formato di

* Texas Instruments

buffer del video. Se progettate una vostra interfaccia video sulla base della visualizzazione alfanumerica 80 per 25, il vostro programa funzionerà sulla maggior parte dei PC e dei compatibili con poche modifiche, se non invariato.

Sfortunatamente, un'alta compatibilità di solito si ottiene sacrificando la velocità. Le routine veloci di output su schermo generalmente sfruttano le idiosincrasie dell'hardware quindi diventano meno trasportabili su diversi dispositivi hardware di visualizzazione rispetto alle routine che si basano su chiamate di sistema operativo o di BIOS più lente ma più universali. Questo compromesso vale per quasi tutte le routine di output su schermo che scriverete.

Rappresentazione dei dati alfanumerici

Tutti i sottosistemi di visualizzazione dei PC e dei PS/2 IBM utilizzano lo stesso formato per la memorizzazione dei dati alfanumerici all'interno del buffer del video. Ogni carattere viene rappresentato da una semplice struttura di dati da 2 byte (vedere Figura 23). I caratteri vengono memorizzati nel buffer in una sequenza lineare che percorre lo schermo orizzontalmente e verticalmente (vedere Figura 24).

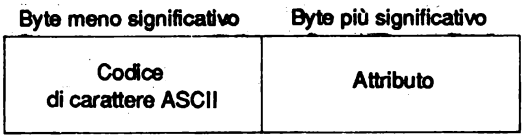


Figura 23. *Mappatura di un carattere alfanumerico e dell'attributo in una parola di 16 bit.*

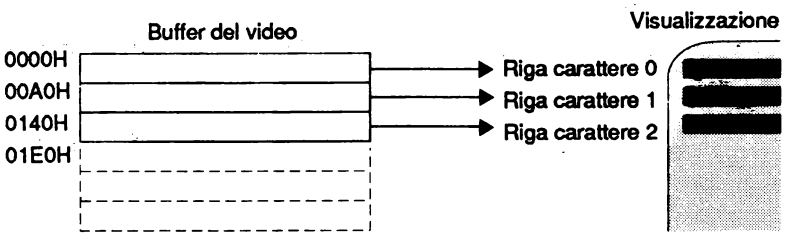


Figura 24. *Mappa del buffer del video nelle modalità alfanumeriche 80 per 25.*

Un generatore di caratteri hardware converte sullo schermo ogni codice di carattere nella corretta configurazione di punti. Contemporaneamente, la circuiteria del decodificatore di attributo genera l'attributo corretto (colore, intensità o luminosità, lampeggiamento, eccetera) per ogni carattere. Dal momento che ogni codice di carattere del buffer

del video è corredato di un byte di attributo, potete controllare individualmente sullo schermo gli attributi di visualizzazione di ogni carattere.

Il generatore di caratteri hardware visualizza ogni carattere alfanumerico all'interno di una matrice rettangolare di pixel. All'interno di questa matrice di carattere, il carattere stesso è composto da una serie di pixel di primo piano. I colori dei pixel di primo piano e di sfondo del carattere vengono specificati dai semi-byte meno significativi e più significativi del corrispondente byte di attributo.

Per visualizzare un carattere, si memorizza il codice ASCII relativo e l'attributo nella locazione corretta all'interno del buffer del video. Grazie allo schema di mappatura lineare, è possibile calcolare facilmente l'indirizzo di buffer di una particolare locazione di schermo. La formula generale è:

```
offset = ((riga x larghezza) + colonna) x 2
```

In questa formula, *larghezza* corrisponde al numero di caratteri di ogni riga. Il fattore 2 è indicato in quanto ogni carattere richiede 2 byte (una parola da 16 bit) di memorizzazione nel buffer del video. I valori di *riga* e di *colonna* sono su base zero a partire dall'angolo superiore sinistro dello schermo (il carattere posto nell'angolo superiore sinistro si trova alla riga 0, colonna 0).

Se esaminate il contenuto del buffer del video, potete vedere come i dati corrispondano ai caratteri sullo schermo (vedere Figura 25). Si noti come ogni codice di carattere sia seguito dal proprio byte di attributo (tutti i byte di attributo nell'area del buffer del video mostrata nella Figura 25 hanno il valore 07H).

```
            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
B000:0000  43 07 68 07 61 07 72 07 61 07 63 07 74 07 65 07  C.h.a.r.a.c.t.e.
B000:0010  72 07 20 07 72 07 6F 07 77 07 20 07 30 07 30 07  r. .r.o.w. .0.0.
B000:0020  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0030  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0040  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0050  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0060  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0070  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0080  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0090  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:00A0  43 07 68 07 61 07 72 07 61 07 63 07 74 07 65 07  C.h.a.r.a.c.t.e.
B000:00B0  72 07 20 07 72 07 6F 07 77 07 20 07 30 07 31 07  r. .r.o.w. .0.1.
B000:00C0  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:00D0  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:00E0  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:00F0  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0100  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0110  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0120  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
B000:0130  00 07 00 07 00 07 00 07 00 07 00 07 00 07 00 07  ................
```

Figura 25. *Stampa esadecimale di un buffer alfanumerico del video.*

Attributi

Sebbene tutti i sottosistemi di visualizzazione dei PC e dei PS/2 IBM utilizzino la stessa configurazione di codici di carattere e di byte di attributo alternati per rappresentare dati alfanumerici, il modo in cui viene interpretato il byte di attributo varia. In generale, il byte di attributo viene formattato come due semi-byte da 4 bit. Il semi-byte meno significativo (i bit da 0 a 3) determina l'attributo del primo piano del carattere, cioè il colore e la luminosità del carattere stesso. Il semi-byte più significativo (i bit da 4 a 7) indica l'attributo di sfondo del carattere, sebbene il bit 7 possa in alcuni casi controllare anche il lampeggiamento.

Gli attributi di primo piano e di sfondo da 4 bit vengono infine decodificati in una serie di segnali che controllano il monitor. Nel più semplice dei casi, su CGA, i quattro bit corrispondono direttamente ai tre segnali dei colori e al segnale della luminosità. Lo schema di decodifica su altri sottosistemi di visualizzazione può essere complesso, come su EGA, MCGA, VGA e sulla scheda InColor, o relativamente semplice come su MDA.

MDA

Sebbene potreste specificare uno dei 16 (2^4) attributi sia per l'attributo di sfondo sia per l'attributo di primo piano, l'MDA riconosce solo determinate combinazioni (vedere Figura 26). Ciononostante, potete generare un'ampia gamma di attributi di carattere combinando in modo creativo luminosità, lampeggiamento e sottolineatura. Potete anche scambiare i normali attributi di primo piano e di sfondo per ottenere l'"inversione video", caratteri scuri su sfondo di luminosità normale.

Non sottolineato

		Primo piano		
	Nero	**Attenuato***	**Intensità normale**	**Evidenziato**
Sfondo				
Nero	00	**	07	0F
Attenuato*	**	88	87	8F
Normale	70	78	**	**
Evidenziato	F0	F8	**	**

* = non visualizzabile da tutti i monitor
** = non disponibile

Sottolineato

	Primo piano	
	Intensità normale	**Evidenziato**
Sfondo		
Nero	01	09
Attenuato*	81	89

* = non visualizzabile da tutti i monitor

Figura 26. *Combinazioni di attributi di primo piano e di sfondo dell'MDA (valori in esadecimale). I valori di attributo non contenuti in questa tabella producono sempre una delle combinazioni mostrate.*

Sull'MDA, oltre che su tutti gli altri dispositivi hardware di visualizzazione IBM, il bit 7 di ogni byte di attributo di carattere ha due scopi. Per default, questo bit determina se il carattere lampeggerà al momento della visualizzazione; l'impostazione a 1 del bit provocherà il lampeggiamento del carattere relativo. Il bit 7 controlla il lampeggiamento in quanto il bit 5 (il bit di abilitazione lampeggiamento) del registro di controllo modalità (3B8H) del CRT dell'MDA è impostato a 1 dal BIOS del video all'accensione del computer.

Se il bit di abilitazione lampeggiamento è a 0, il bit 7 del byte di attributo non controllerà più il lampeggiamento (vedere Listato 11), ma verrà interpretato come bit della luminosità per l'attributo dello sfondo. Quando il bit 7 viene impostato in un byte di attributo di carattere, la luminosità dell'attributo dello sfondo del carattere viene aumentata, ciò significa che il verde normale diventa verde evidenziato e il nero diventa verde scuro. Di conseguenza, per ottenere tutte le possibili combinazioni degli attributi monocromatici elencati nella Figura 26, dovrete impostare a 0 il bit di abilitazione lampeggiamento.

```
mov     ax,40h
mov     es,ax        ; ES := segmento dati BIOS video
mov     dx,es:[63h]; DX := 3B4h (MDA) o
                     ;  3D4h (CGA) da ADDR_6845
add     dl,4         ; DX := 3x8h (registro controllo modalità
                     ;  CRT)
mov     al,es:[65h]; AL := valore corrente registro
                     ; (CRT_MODE_SET)
and     al,11011111b    ; azzera bit 5
out     dx,al        ; aggiorna il registro
mov     es:[65h],al; aggiorna l'area dati del BIOS
```

Listato 11. *Azzeramento del bit di abilitazione lampeggiamento su MDA o CGA.*

Il valore del bit di abilitazione lampeggiamento del registro di controllo modalità influenza l'interpretazione del bit 7 di tutti i byte di attributo, quindi non è possibile visualizzare contemporaneamente caratteri lampeggianti e caratteri con sfondo evidenziato. Dovete decidere quale attributo è più utile nel vostro programma ed impostare di conseguenza il bit di abilitazione lampeggiamento.

Tutti i sottosistemi di visualizzazione dei PC e dei PS/2 IBM, compreso l'MDA, producono il lampeggiamento dei caratteri alfanumerici sostituendo l'attributo di sfondo con l'attributo di primo piano ad un ritmo di circa due volte al secondo. L'effetto risultante è che ogni carattere lampeggiante si alterna ad un carattere di spaziatura.

Se riempite lo schermo di caratteri lampeggianti, l'effetto complessivo potrebbe essere sconcertante in quanto lo schermo viene cancellato e ripristinato due volte al secondo. Tuttavia se il vostro scopo è quello di attirare l'attenzione sullo schermo, l'uso dell'attributo di lampeggiamento può essere molto efficace.

CONSIGLIO

Se utilizzate l'attributo di sottolineatura (attributo 1 o 9 di primo piano) su un portatile Compaq, non vedrete i caratteri sottolineati. Ciò accade perché il portatile Compaq decodifica i valori di attributo in 16 gradazioni di verde progressivamente più luminose; i valori di 1 e 9 dell'attributo di sottolineatura appaiono perciò come gradazioni di verde.

CONSIGLIO

Potrebbe sorprendere che alcuni MDA IBM generano output a colori oltre che output monocromatico. Naturalmente, il video monocromatico verde dell'MDA usa solo due segnali per controllare gli attributi (visualizzazione attiva/disattiva ed evidenziazione attiva/disattiva); esso ignora qualsiasi segnale di visualizzazione colore. Tuttavia, un video a colori che può utilizzare i segnali di sincronizzazione orizzontale di 16,257 MHz e di sincronizzazione verticale di 50 Hz dell'MDA visualizzerà otto colori (con due varianti di luminosità) quando viene collegato ad alcuni MDA (ma non a tutti). Sfortunatamente, non potrete mai essere certi di quale MDA si trasformerà in un adattatore a colori.

HGC

L'HGC e l'HGC+ emulano perfettamente la modalità alfanumerica monocromatica dell'MDA. I programmi scritti per MDA funzionano su entrambi questi adattatori senza apportare modifiche.

CGA

Il CGA applica lo stesso schema di attributo di primo piano e sfondo dell'MDA. Tuttavia, la circuiteria del decodificatore di attributo del CGA riconosce tutte le 16 possibili combinazioni dei quattro bit di ogni semi-byte del byte di attributo. Per ogni carattere dello schermo, potete specificare separatamente uno dei 16 colori per il primo piano e lo sfondo.

I colori disponibili sono semplici combinazioni dei colori primari rosso, verde e blu. Ogni bit di ogni semi-byte del byte di attributo corrisponde ad un segnale che il CGA fornisce al monitor (vedere Figura 27). I tre bit meno significativi di ogni semi-byte corrispondono ai segnali di rosso (R), verde (G) e blu (B). Le otto possibili combinazioni producono una gamma di colori comprendenti il rosso, il verde, il blu e i loro colori intermedi (vedere Figura 28).

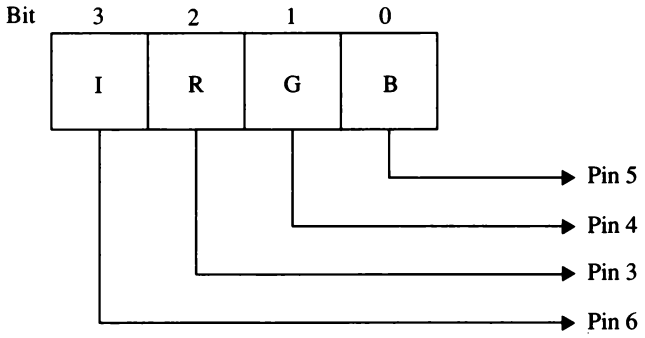


Figura 27. *Attributi e segnali guida del colore del monitor del CGA. Il numero dei pin si riferisce al connettore a 9 pin del CGA.*

Colore	Binario (IRGB)	Esadecimale
Nero	0000	00
Blu	0001	01
Verde	0010	02
Ciano	0011	03
Rosso	0100	04
Viola	0101	05
Giallo (marrone)	0110	06
Bianco	0111	07
Nero (grigio)	1000	08
Blu evidenziato	1001	09
Verde evidenziato	1010	0A
Ciano evidenziato	1011	0B

(continua)

Colore	Binario (IRGB)	Esadecimale
Rosso evidenziato	1100	0C
Viola evidenziato	1101	0D
Giallo evidenziato	1110	0E
Bianco evidenziato	1111	0F

Figura 28. *Attributi di visualizzazione del CGA.*

L'impostazione del bit 3 del byte di attributo (il bit della luminosità all'interno del semi-byte del primo piano) visualizza il colore indicato nei bit R, G e B (i bit da 0 a 2) con una maggiore luminosità. In ogni caso, come sull'MDA, il bit più significativo (il bit 7) di ogni byte di attributo controlla o l'evidenziazione dello sfondo o il lampeggiamento. Anche in questo caso, ciò che l'attributo visualizza dipende dallo stato di un bit nel registro di controllo.

Il bit 5 del registro di controllo modalità del CGA (porta di I/O 3D8H) è un bit di abilitazione lampeggiamento analogo al bit 5 del registro di controllo del CRT dell'MDA. Quando impostate a 0 il bit di abilitazione lampeggiamento, il bit 7 di un byte di attributo di carattere, specificate che il colore di sfondo indicato nei bit da 4 a 6 dovrebbe essere visualizzato con una luminosità maggiore. Quando impostate a 1 il bit di abilitazione lampeggiamento, verranno visualizzati solo i colori di sfondo non evidenziati, ma i caratteri i cui byte di attributo hanno il bit 7 impostato a 1 lampeggeranno.

Il bit di abilitazione lampeggiamento è impostato a 1 ogni qualvolta richiamate il BIOS ROM per selezionare una modalità di visualizzazione alfanumerica. Per default, quindi, il bit 7 del byte di attributo di ogni carattere controlla il lampeggiamento e non l'evidenziazione dello sfondo. Dovrete azzerare il bit di abilitazione lampeggiamento per visualizzare caratteri con colori di sfondo evidenziati.

Molti monitor compatibili CGA riescono ad ottenere di più dai 16 colori disponibili (8 non evidenziati e 8 evidenziati) visualizzando il giallo non evidenziato come marrone e il nero evidenziato come grigio. Sfortunatamente, la possibilità di produrre ciò non può essere determinata da un programma. Prestate attenzione a non visualizzare, ad esempio, caratteri grigi su sfondo nero con un CGA, in quanto queste combinazioni di colore risultano invisibili su alcuni monitor a colori.

EGA

Nelle modalità alfanumeriche a 16 colori, l'EGA usa lo stesso formato di byte di attributo del CGA. Tuttavia, i valori di primo piano e di sfondo a 4 bit non corrispondono direttamente ai colori visualizzati. Ogni valore da 4 bit è invece mascherato con i quattro bit meno significativi del registro di abilitazione piano di colore (12H) del controller di attributo; il valore risultante di 4 bit indica uno dei 16 registri di tavolozza dell'EGA (vedere Figura 29). Ogni bit del valore a 6 bit del colore contenuto nel registro di tavolozza

indicato corrisponde ad uno dei sei segnali RGB che controllano il monitor (vedere Figura 30).

Un monitor a colori compatibile EGA viene controllato da sei segnali di colore: tre primari (evidenziati) e tre secondari (non evidenziati). Tutte le 64 combinazioni di questi sei segnali appaiono come diversi colori e/o intensità. Con un monitor a colori da 200 linee (o nelle modalità a 200 linee su un monitor compatibile EGA) i bit 0, 1 e 2 controllano i segnali del colore, mentre il bit 4 controlla il segnale della luminosità.

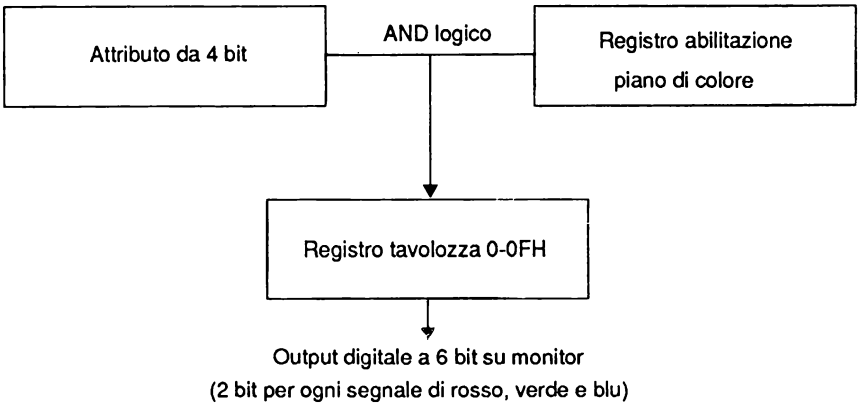


Figura 29. *Attributi e colori su EGA.*

Il metodo utilizzato dall'EGA per generare indirettamente i colori tramite i registri di tavolozza è più complesso dello schema diretto del CGA, ma l'EGA risulta più flessibile. Potete selezionare i colori di primo piano e di sfondo individualmente per ogni carattere, ma nel contempo potete produrre delle modifiche globali del colore aggiornando il valore di un particolare registro tavolozza.

Il bit più significativo del byte di attributo di ogni carattere può controllare o il lampeggiamento o l'intensità dello sfondo, esattamente come su MDA e su CGA. Il bit 3 del registro di controllo modalità del controller di attributo dell'EGA (il registro 10H alla porta di I/O 3C0H) è il bit di abilitazione lampeggiamento. L'impostazione a 1 di questo bit abilita il lampeggiamento, quindi solo i 3 bit meno significativi del semi-byte dello sfondo (i bit da 4 a 6 del byte di attributo) indicano i registri di tavolozza. Di conseguenza, quando viene abilitato il lampeggiamento, potete far riferimento solo ai primi otto registri di tavolozza per selezionare il colore di sfondo di un carattere. L'impostazione a 0 del bit di abilitazione lampeggiamento disabilita il lampeggiamento, rendendo disponibili tutti i 16 registri di tavolozza per i colori di sfondo (vedere Listato 12).

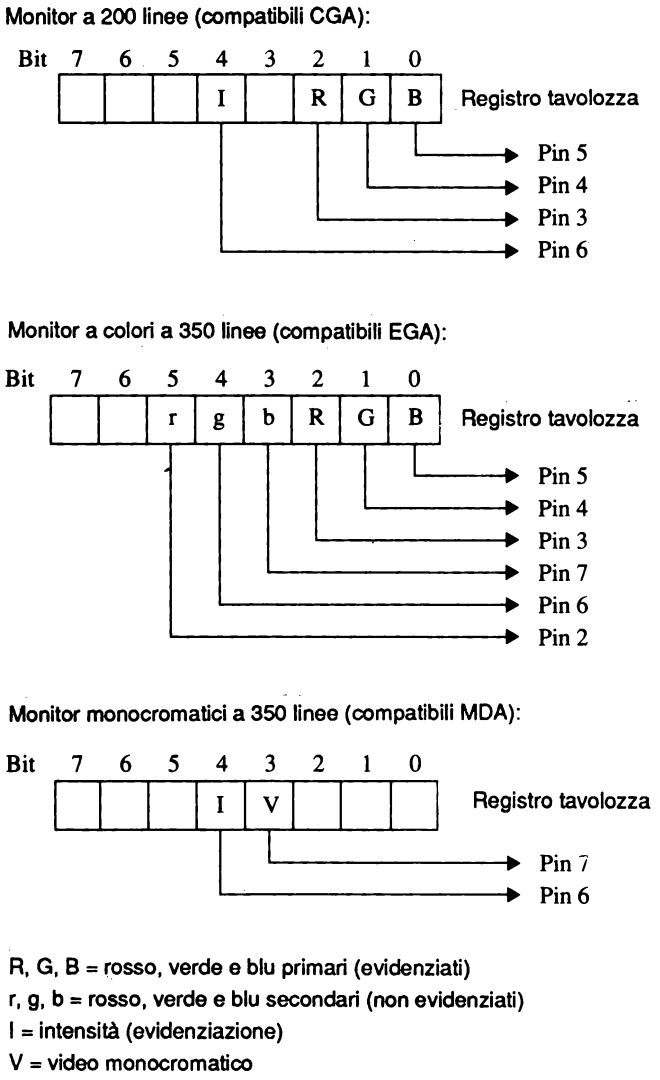


Figura 30. *Valori del registro tavolozza dell'EGA e segnali di controllo colore del monitor corrispondenti. I numeri di pin si riferiscono al connettore a 9 pin dell'EGA.*

```
mov     bl,0        ; BL := valore per il bit di abilitazione
                    ;  lampeggiamento
mov     ax,1003h    ; AH := numero funzione INT 10H
                    ; AL := numero sottofunzione
int     10h
```

Listato 12. *Impostazione ed azzeramento del bit di abilitazione lampeggiamento su MCGA, EGA o VGA.*

Quando selezionate una modalità di visualizzazione alfanumerica utilizzando il BIOS dell'EGA, i registri di tavolozza vengono caricati con i valori di default che corrispondono ai colori disponibili su CGA. I valori di colore dei secondi otto registri di tavolozza sono le versioni evidenziate di quelli dei primi otto. Di conseguenza, se trattate semplicemente il bit 7 del byte di attributo come bit di "evidenziazione o di lampeggiamento", il vostro programma funzionerà sia su EGA sia su CGA.

Potete aggiornare il contenuto di qualsiasi registro di tavolozza o direttamente o tramite la funzione 10H di INT 10H (vedere Listato 13). L'uso della routine BIOS risulta più comodo ed evita la necessità di scrivere codici dipendenti dall'hardware. Inoltre, la routine del BIOS può anche caricare contemporaneamente tutti i 16 registri di tavolozza, avendo stabilito in precedenza una tabella di valori di colore (vedere l'Appendice A). Ciononostante, potreste dover programmare direttamente i registri di tavolozza per produrre modifiche di colore molto rapide come quelle richieste in alcuni tipi di animazione.

```
; aggiornamento diretto di un registro di tavolozza:

            mov     ax,40h
            mov     es,ax       ; ES := segmento dati del BIOS del video
            mov     dx,es:[63h] ; DX := registro indirizzo del CRTC (3x4h)
            add     dl,6        ; DX := registro di stato (3xAh)
            push    dx          ; mantiene questo valore
            cli
            in      al,dx       ; azzera flip-flop indirizzo controller
                                ;  dell'attributo
            mov     dl,0C0h     ; DX := 3C0h
            mov     al,PaletteRegNumber
            out     dx,al       ; aggiorna un registro di tavolozza
            mov     al,PaletteRegValue
            out     dx,al
            pop     dx          ; DX := porta registro di stato
            in      al,dx       ; azzera flip-flop
            mov     dl,0C0h
            mov     al,20h
            out     dx,al       ; imposta bit 5 del registro di
                                ;  indirizzo del controller di attributo
            sti

; aggiornamento di un registro di tavolozza utilizzando il BIOS del video

            mov     bl,PaletteRegNumber
            mov     bh,PaletteRegValue
            mov     ax,1000h    ; AH := numero funzione INT 10H
                                  AL := numero sottofunzione
            int 10h
```

Listato 13. *Programmazione del registro di tavolozza su EGA o VGA.*

Nella modalità alfanumerica monocromatica, l'EGA emula gli attributi di visualizzazione monocromatica dell'MDA. Il BIOS del video inizializza i registri di tavolozza con

i valori che corrispondono agli attributi dell'MDA (vedere Figura 31). Il bit 3 determina se i pixel sono on o off e il bit 4 (se impostato oltre al bit 3) produce una visualizzazione evidenziata. L'attributo di sottolineatura viene generato ogni qualvolta l'attributo di primo piano di un carattere è 1 o 9, a prescindere dal valore del corrispondente registro di tavolozza.

Valore	Attributo
0	Nero
8	Non evidenziato
10H	Attenuato
18H	Evidenziato

Figura 31. *Valori di attributo alfanumerico monocromatico per i registri di tavolozza dell'EGA.*

CONSIGLIO

L'EGA genera anche un attributo di sottolineatura nelle modalità alfanumeriche a 16 colori quando l'attributo di primo piano è 1 o 9 e l'attributo di sfondo è 0 o 8. In ogni caso, non vedrete normalmente la sottolineatura nelle modalità a 16 colori in quanto il valore di default del BIOS di visualizzazione per il registro di locazione sottolineatura del CRTC (14H) è 1FH. Questo valore è maggiore del numero di linee di scansione visualizzate normalmente per caratteri alfanumerici, quindi la sottolineatura non appare.
Potete generare caratteri sottolineati nelle modalità a 16 colori memorizzando un valore visualizzabile nel registro di locazione sottolineatura. Naturalmente, solo i caratteri con attributi di 1, 9, 81H o 89H appariranno sottolineati, ma potete modificare i valori nei corrispondenti registri di tavolozza per produrre caratteri sottolineati di qualsiasi colore desiderato.

Scheda InColor

La scheda InColor può decodificare gli attributi alfanumerici in diversi modi. La scheda ha una serie di 16 registri di tavolozza, le cui funzioni sono analoghe a quelle dei registri di tavolozza del controller di attributo dell'EGA. La scheda InColor può però essere configurata dal vostro programma in modo che aggiri i registri di tavolozza e decodifichi gli attributi di primo piano e di sfondo di 4 bit di ogni carattere compatibilmente a MDA o CGA. I bit 4 e 5 del registro di eccezione (17H) controllano l'interpretazione da parte della scheda InColor degli attributi alfanumerici (vedere Figura 32). Il bit 5 determina se la scheda InColor visualizzerà attributi monocromatici (come su MDA) o attri-

buti di colore (come su CGA o EGA). Il bit 4 abilita la mappatura di attributo tramite i registri di tavolozza.

Quando la scheda InColor riceve l'alimentazione, il bit 5 del registro di eccezione ha il valore 1 e il bit 4 ha il valore 0. Di conseguenza, per default, la scheda interpreta gli attributi come farebbe l'MDA. Tuttavia, se impostate a 0 sia il bit 5 sia il bit 4 (vedere Listato 14), gli attributi alfanumerici specificheranno la stessa serie di 16 colori esattamente come su CGA (fare riferimento alla Figura 28).

```
mov     ax,0017h    ; AH bit 5 := 0 (disabilita
                    ;  attributi monocromatici)
                    ; Ah bit 4 := 0 (disabilita tavolozza)
                    ; AH bit da 0 a 3 := 0 (colore di default
                    ;  del cursore)
                    ; AL := 17h (numero registro di eccezione)
mov     dx,3B4h     ; DX := porta di I/O
out     dx,ax
```

Listato 14. *Programmazione del registro di eccezione della scheda InColor.*

Bit 5	Bit 4	Emulazione attributo
0	0	CGA
0	1	EGA
1	0	MDA
1	1	MDA mappata tramite registri di tavolozza

Figura 32. *Controllo del registro di eccezione degli attributi su scheda InColor Hercules.*

L'impostazione ad 1 del bit 4 provoca il collegamento degli attributi ai registri di tavolozza della scheda, a prescindere dal valore del bit 5. Di conseguenza, se il bit 4 è a 1 e il bit 5 è a 0, la scheda InColor interpreta gli attributi come li interpreta l'EGA. Se il bit 4 è a 1 e il bit 5 è a 1, invece, la scheda collega gli attributi di primo piano e di sfondo di ogni carattere solo ai registri di tavolozza che corrispondono a valori di attributo monocromatico validi. In questo caso, gli attributi "nero", "attenuato", "non evidenziato" ed "evidenziato" selezionano rispettivamente i registri di tavolozza 00H, 08H, 07H e 0FH.

Il bit 5 del registro di controllo modalità del CRT a 3B8H è il bit di abilitazione lampeggiamento. Questo bit controlla l'evidenziazione dello sfondo a prescindere dai valori dei bit 4 e 5 del registro di eccezione. Tuttavia, i caratteri vengono fatti lampeggiare solo se il bit 5 del registro di eccezione è a 1 (attributi compatibili MDA); i caratteri non lampeggiano se il bit 5 del registro di eccezione è a 0 (attributi compatibili CGA), a prescindere dall'impostazione del bit di abilitazione lampeggiamento.

Non viene fornito alcun supporto di BIOS del video per i registri di tavolozza della scheda InColor. Il vostro programma dovrà perciò aggiornare la tavolozza memorizzando direttamente i valori nei registri di tavolozza. Il Listato 15 è un esempio di come pro-

cedere. La lettura di I/O iniziale (*IN AL,DX*) del registro di tavolozza (1CH) azzera un indice interno che punta il primo dei 16 registri di tavolozza interni. Ogni operazione di I/O di scrittura successiva (*OUT DX,AL*) aggiorna un registro di tavolozza interno ed incrementa l'indice interno in modo che punti al successivo registro di tavolozza, producendo così il caricamento di tutti i 16 registri tramite l'esecuzione di un semplice loop.

CONSIGLIO

Dal momento che gli attributi monocromatici possono essere mappati tramite i registri di tavolozza, potete assegnare fino a quattro diversi colori ai programmi monocromatici che girano su scheda InColor. Ciò viene realizzato impostando a 1 i bit 4 e 5 del registro di eccezione e aggiornando i registri di tavolozza 00H, 08H, 07H e 0FH con i colori desiderati.

```
            mov     dx,3B4h         ; DX := registro di indirizzo del CRTC
            mov     al,1Ch          ; AL := 1Ch (numero di registro di
                                    ;  tavolozza)
            out     dx,al
            inc     dx              ; DX := 3B5h
            in      al,dx           ; azzera indice registro di tavolozza

            mov     si,offset PaletteTable ; DS:SI -> PaletteTable
            mov     cx,16           ; CX := numero dei registri di
                                    ;  tavolozza
L01:        lodsb                   ; AL := byte successivo dalla tabella
            out     dx,al           ; aggiorna successivo registro di
                                    ;  tavolozza
            loop    L01
            .
            .
            .
PaletteTable db     00h,01h,02h,03h,04h,05h,06h,07h
                                ; registri tavolozza  da 0-7
            db      38h,39h,3Ah,3Bh,3Ch,3Dh,3Eh,3Fh
                                ; registri tavolozza da 8-0Fh
```

Listato 15 *Programmazione del registro di tavolozza della scheda inColor.*

Sulla scheda InColor, i colori del cursore e della sottolineatura sono indipendenti dai colori di primo piano dei caratteri contenuti nel buffer del video. Il colore del cursore viene specificato nei bit da 0 a 3 del registro di eccezione, e il valore del colore di sottolineatura viene specificato dai bit da 4 a 7 del registro di sottolineatura (registro 15H del CRTC). Quando la scheda InColor sta visualizzando gli attributi MDA (cioè, quando il bit 5 del registro di eccezione è impostato a 1), potete specificare solo i tre bit meno significativi dei colori del cursore e della sottolineatura; il bit più significativo di questi valori di colore deriva dall'attributo di primo piano del carattere dove viene visualizzato il cursore o la sottolineatura.

Quando la mappatura di tavolozza è abilitata (bit 4 del registro di eccezione impostato a 1), sia il valore di colore del cursore sia quello della sottolineatura selezionano i registri di tavolozza. Quando la mappatura di tavolozza è disabilitata, i valori di colore del cursore e della sottolineatura vengono visualizzati utilizzando i soliti colori CGA. Inoltre, se specificate un valore di 0 per il colore di sottolineatura o di cursore, la scheda InColor usa al posto di questo valore il valore 7.

MCGA

I componenti del sottosistema di visualizzazione del modello 30 del PS/2 che trasformano i dati di attributo in segnali per video a colori sono il formattatore video e il convertitore digitale-analogico (DAC). La circuiteria del formattatore video decodifica gli attributi e genera un output digitale a 8 bit che viene trasmesso al DAC del video. Il DAC converte l'output a 8 bit proveniente dal formattatore video in tre segnali di colore analogici utilizzando gli 8 bit per selezionare uno dei 256 registri di colore del DAC. Ogni registro di colore del DAC ha un'ampiezza di 18 bit, comprendendo i tre valori da 6 bit per il rosso, il verde e il blu (vedere la Figura 34). Il DAC converte ogni valore da 6 bit in un segnale analogico con il valore massimo (3FH) corrispondente al segnale di massima intensità.

Nelle modalità alfanumeriche, i quattro bit meno significativi dell'output digitale a 8 bit del formattatore video derivano dai byte di attributo, mentre i quattro bit più significativi sono sempre a 0 (vedere Figura 33). Di conseguenza, solo i primi 16 dei registri di colore del DAC del video vengono utilizzati nelle modalità alfanumeriche dell'MCGA. Ai restanti 240 registri è possibile accedere solo nella modalità grafica a 256 colori 320 per 200 (vedere Capitolo 4). Quando un MCGA viene collegato ad un monitor a colori, il BIOS del video inizializza i primi 16 registri del colore del DAC del video con gli stessi colori del CGA.

CONSIGLIO

Il valore del registro maschera del DAC del video (porta di I/O 3C6H) maschera il valore di 8 bit trasmesso al DAC del video. Il valore del registro maschera è impostato a 0FFH dalle routine di inizializzazione del BIOS del video in modo che sia possibile accedere a tutti i 256 registri del colore del DAC del video. La documentazione tecnica dell'IBM consiglia di non modificare questo valore.

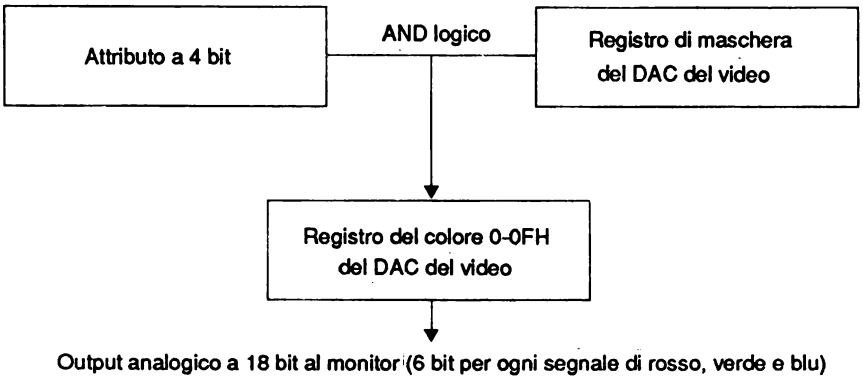


Figura 33. *Attributi e colori su MCGA (il valore del registro di maschera del DAC del video dovrebbe normalmente essere 0FFH).*

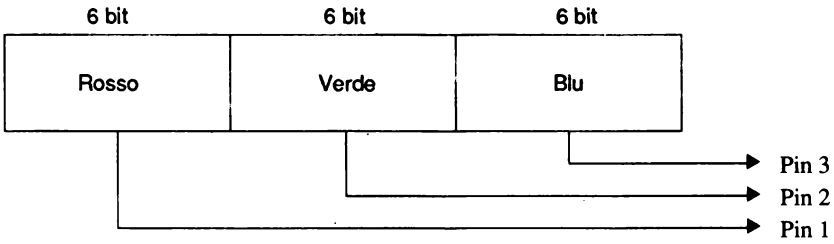


Figura 34. *Valori del registro del colore del DAC del video e segnali di controllo del colore del monitor. Il numero dei pin si riferisce al connettore a 15 pin dell'MCGA.*

A differenza dell'EGA, l'MCGA con un monitor monocromatico non emula gli attributi dell'MDA. Al contrario, i 16 valori di default del registro di colore del DAC del video sono costituiti da quattro gruppi di quattro gradazioni di grigio. Ogni gruppo viene visualizzato con un'intensità maggiore del precedente. All'interno di ogni gruppo, l'intensità aumenta dal valore di attributo inferiore al valore superiore. Di conseguenza, i valori di attributo da 0 a 3 costituiscono una gamma di quattro gradazioni di grigio, i valori da 4 a 7 una seconda gamma con un'intensità in qualche modo maggiore e i valori da 8 a 0BH e da 0CH a 0FH una terza e una quarta gamma che presentano un'intensità ancora superiore.

CONSIGLIO

Al posto di questa configurazione di default di scala di grigio monocromatica dell'MCGA, potreste voler utilizzare i valori di scala di grigio che aumentano uniformemente con l'aumentare dei valori di attributo. Il codice contenuto nel Listato 14 carica i registri del DAC del video con appropriati valori per questa gamma di scala di grigi.

```
            mov     bx,0Fh          ; BX := primo numero di registro
                                    ;  del colore del DAC del video
            mov     di,offset VDACTable  ; DS:DI ->tabella

L01:        mov     dh,[bx+di]      ; DH := valore del rosso
            mov     ch,dh
            mov     cl,dh           ; i valori del verde e del blu sono
                                    ;  identici
            mov     ax,1010h        ; AH := numero di funzione INT 10H
                                    ; AL := numero di sottofunzione
            int     10h
            dec     bx
            jns     L01             ; loop dal registro 0FH al registro 0
            .
            .
            .
VDACTable           db    00h,05h,08h,0Bh,0Eh,11h,14h,18h
                    db    1Ch,20h,24h,28h,2Dh,32h,38h,3Fh
```

Listato 14. *Caricamento di una tavolozza alternativa di scala di grigi monocromatica MCGA.*

VGA

In generale, la VGA emula esattamente la decodifica di attributo alfanumerico dell'EGA. Tuttavia, la VGA ha sia il DAC del video sia una serie di 16 registri di tavolozza di controller di attributo. Ogni valore di registro di tavolozza seleziona uno dei 256 registri del colore del DAC del video. Il valore selezionato del registro del colore del DAC del video determina il colore visualizzato.

A seconda del valore del bit 7 del registro di controllo modalità del controller di attributo, potete utilizzare il valore del registro di tavolozza per selezionare un registro del colore del DAC del video in due modi. Quando il bit 7 è impostato a 0, il controller di attributo combina il valore del registro di tavolozza da 6 bit con i bit 2 e 3 del proprio registro di selezione colore (14H) per produrre un valore a 8 bit che seleziona un registro del colore del DAC del video (vedere Figura 35). In alternativa, quando il bit 7 è impostato a 1, solo i quattro bit meno significativi di ogni registro di tavolozza assumono un significato. Il controller di attributo ricava gli altri quattro bit del valore a 8 bit dai bit da 0 a 3 del registro di selezione colore (vedere Figura 36).

Nel primo caso (quando il bit 7 del registro di controllo modalità è impostato a 0), i registri di tavolozza a 6 bit vengono utilizzati per selezionare uno dei quattro gruppi di

64 registri del colore del DAC del video e i bit 2 e 3 del registro di selezione colore determinano quale gruppo di registri del colore viené utilizzato. Nel secondo caso (quando il bit 7 del registro di controllo modalità è impostato a 1), ogni valore di registro di tavolozza seleziona uno dei 16 gruppi di 16 registri del colore del DAC del video e i bit da 0 a 3 del registro di selezione colore specificano uno dei 16 gruppi di registri del colore del DAC

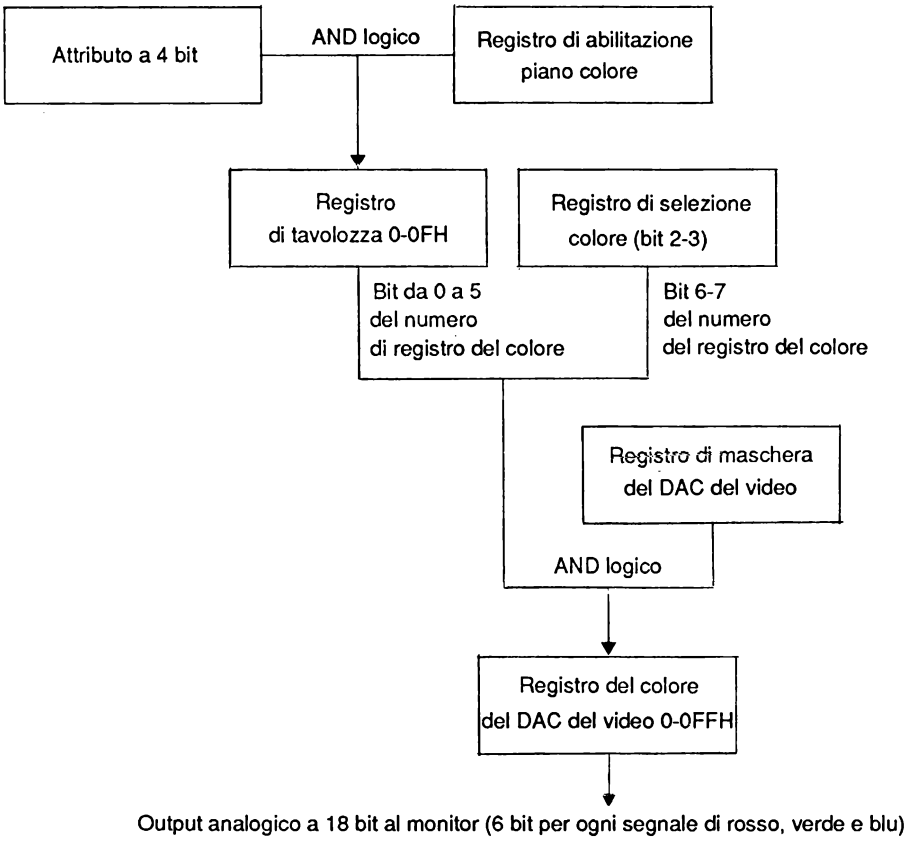


Figura 35. *Attributi e colori su VGA (quando il bit 7 del registro di controllo modalità del controller di attributo è impostato a 0).*

Questa ulteriore possibilità, permessa dall'uso combinato dei registri di tavolozza e dei registri del colore del DAC del video, rende semplice il passaggio da una tavolozza all'altra, dal momento che potete selezionare una qualsiasi delle 16 diverse tvolozze da 16 colori semplicemente modificando il valore del registro di selezione colore del controller di attributo. Se memorizzate 16 tavolozze di intensità gradualmente crescente nei registri del colore del DAC, potete accentuare i caratteri sullo schermo aumentando e di-

minuendo ciclicamente la loro intensità. Questo effetto è più sofisticato del semplice lampeggiamento del carattere, in particolare quando viene applicato ad un'area estesa dello schermo..

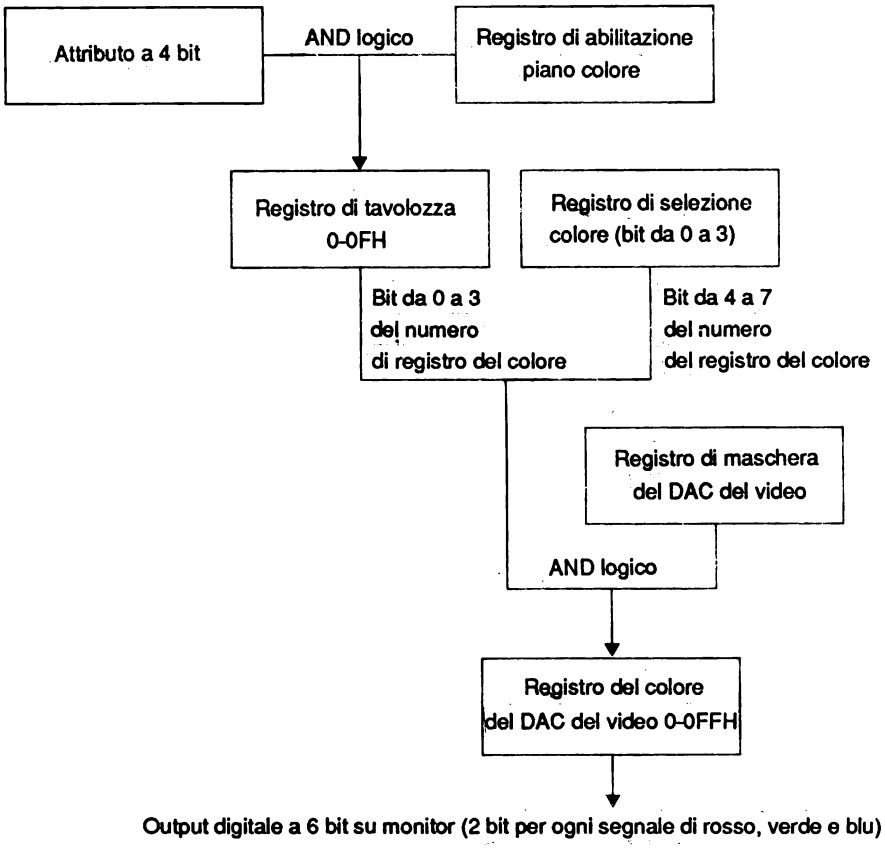


Figura 36. *Attributi e colori su VGA (quando il bit 7 del registro di controllo modalità del controller di attributo è impostato a 1*

CONSIGLIO

Quando la VGA emula la modalità alfanumerica a 16 colori 80 per 25 su uno schermo monocromatico, la tavolozza è costituita dagli stessi quattro gruppi di quattro gradazioni di grigio come avviene nella tavolozza corrispondente su MCGA. Come su MCGA, potete creare una tavolozza a scala di grigi con intensità gradualmente crescenti. Il Listato 17 illustra come procedere. Si noti come i registri appropriati del DAC del video vengono selezionati esaminando i valori dei registri di tavolozza del controller di attributo.

```
            mov     bx,0Fh      ; BX := primo numero di registro di
                                ;  tavolozza
            mov     di,offset VDACTable ; DS:DI -tabella

L01:        mov     dh,[bx+di]  ; DH := valore del rosso
            mov     ch,dh
            mov     cl,dh       ; i valori del verde e del blu sono identici
            push    bx          ; mantiene numero di registro di tavolozza
            mov     ax,1007h    ; AH := numero di funzione INT 10H
                                ; AL := numero di sottofunzione
                                ;  (legge il registro di tavolozza)
            int     10h         ; BH := valore del registro di tavolozza
            mov     bl,bh
            xor     bh,bh       ; BX := numero registro colore desiderato
                                ;  del DAC del video
            mov     ax,1010h    ; AH := numero di funzione INT 10H
                                ; AL := numero di sottofunzione
            int     10h
            pop     bx
            dec     bx          ; BX := numero di registro tavolozza
                                ;  successiva
            jns     L01         ; loop dai registri di tavolozza da 0FH a 0


VDACTable           db   00h,05h,08h,0Bh,0Eh,11h,14h,18h
                    db   1Ch,20h,24h,28h,2Dh,32h,38h,3Fh
```

Listato 17. *Caricamento di una tavolozza di gradazione di grigio monocromatica VGA alternativa.*

La VGA emula la modalità alfanumerica monocromatica dell'MDA (la modalità 7 del BIOS del video) sia su monitor a colori sia su monitor monocromatico. I valori del registro di tavolozza del controller di attributo ed il controllo del lampeggiamento e della sottolineatura sono identici a quelli dell'EGA. In questa modalità, i registri del DAC del video corrispondenti ai valori di tavolozza 00H, 07H, 08H e 18H vengono inizializzati con gli appropriati valori di scala di grigi. In questa modalità, i valori dei registri di tavolozza e di DAC del video sono identici indipendentemente dal fatto che si abbia collegato un monitor a colori o monocromatico.

Calcolo della scala dei grigi

Il BIOS dell'MCGA e della VGA contiene della logica che può trasformare i valori di rosso-verde-blu dei registri del DAC del video in corrispondenti valori di scala di grigi. Questa trasformazione viene realizzata calcolando una media ponderata delle componenti di rosso, verde e blu. Per calcolare il valore equivalente nella scala di grigi, il BIOS somma il 30 percento del valore del rosso, il 59 percento del verde e l'11 percento del

blu (queste percentuali rappresentano approssimativamente le intensità visualizzate di rosso, verde e blu allo stato puro). Ad esempio, il colore di default per il registro 02H (ciano) del colore del DAC del video è costituito da tre componenti da 6 bit. Il valore della componente di rosso è 0, la componente di verde è 2AH e la componente di blu è 2AH. Il valore della scala di grigi è quindi 1DH, la somma cioè di:

(0,30 x 0) + (0,59 x 2AH) + (0,11 x 2AH)

La funzione 10H di INT 10H comprende una sottofunzione (AL=1BH) che legge una serie di registri di colore del DAC del video e li aggiorna con valori equivalenti nella scala dei grigi. L'Appendice A contiene un esempio dell'uso di questa funzione del BIOS del video.

Sia su MCGA sia su VGA, la funzione 0 di INT 10H applica per default il calcolo della scala di grigi quando viene collegato un monitor monocromatico. Con un monitor a colori, il calcolo della scala di grigi viene disabilitato per default. Potete abilitare o disabilitare selettivamente il calcolo per default della scala di grigi eseguendo la funzione 12H di INT 10H con BL = 33H.

Colore del bordo

Su CGA, EGA, MCGA e VGA, potete specificare un colore da visualizzare durante gli intervalli di sovrascansione verticale ed orizzontale. Questo colore di sovrascansione o di bordo non viene rappresentato da nessun dato del buffer del video, ma da uno speciale registro di controllo che contiene il valore del colore visualizzato.

CGA

Su CGA, si seleziona il colore del bordo con i quattro bit meno significativi del registro di selezione colore alla porta di I/O 3D9H (vedere Listato 18). I valori di colore sono in relazione a quelli disponibili per gli attributi di carattere: i bit 0, 1 e 2 selezionano il blu, il verde ed il rosso primari e il bit 3 viene interpretato come bit della luminosità.

```
; aggiornamento diretto del registro del colore del CRT (solo CGA)

            mov     ax,40h
            mov     es,ax           ; ES := segmento dati BIOS video
            mov     dx,es:[63h]     ; DX := 3D4H (ADDR_6845)
            add     dl,5            ; DX := 3D9H (registro selezione
                                    ; colore CRT)
            mov     al,es:[66h]     ; AL := valore corrente registro
```

```
                                ; (CRT_PALETTE)
            and     al,11110000b    ; azzera bit da 0 a 3
            or      al,BorderValue  ; aggiorna bit da 0 a 3
            out     dx,al           ; aggiorna il registro
            mov     es:[66h],al     ; aggiorna l'area dati del BIOS

; uso dell'interfaccia del BIOS del video (CGA, EGA, VGA)

            mov     bl,BorderValue
            mov     bh,0            ; BH := numero di sottofunzione
            mov     ah,0Bh          ; AH := numero di funzione INT 10h
            int     10h
```

Listato 18. *Impostazione di un colore di bordo.*

L'uso della funzione 0BH di INT 10H per aggiornare il colore del bordo risulta probabilmente più comodo rispetto alla programmazione diretta del registro di selezione colore. Il codice è più trasportabile e la routine del BIOS salva nella propria area dati di visualizzazione, nel byte a 0040:0066 (CRT_PALETTE), il valore di registro di selezione colore scritto più recentemente. Se non scrivete direttamente sul registro di selezione colore, dovreste aggiornare CRT_PALETTE come nell'esempio del Listato 18.

CONSIGLIO

L'MCGA non genera un bordo colorato, a prescindere dal valore del registro di selezione colore.

EGA e VGA

Su EGA e VGA, il colore di sovrascansione viene specificato dal contenuto del registro 11H del controller di attributo (porta di I/O 3C0H). Potreste scrivere direttamente sulla porta di I/O, ma effettuare l'operazione con una chiamata INT 10H risulta generalmente più semplice. Potete utilizzare il BIOS dell'EGA per aggiornare in due modi il colore di sovrascansione. Potete usare la funzione 0BH di INT 10H o potete includere il colore del bordo come 17esimo ed ultimo elemento della tabella dei colori di registro di tavolozza trasmessa alla funzione 10H di INT 10H (vedere Appendice A).

CONSIGLIO

Su VGA con un monitor monocromatico, gli unici attributi di bordo utili sono 0 (nero), 8 (normale) e 18H (evidenziato).

Le modalità di visualizzazione a 350 linee su EGA presentano degli intervalli di sovrascansione orizzontale e verticale relativamente brevi. Il bordo visualizzato potrebbe essere largo solo 1 o 2 mm, o potrebbe disperdersi sullo schermo durante l'intervallo di ritracciamento orizzontale. Per questo motivo dovreste evitare di impostare un colore di bordo in una delle modalità a 350 linee dell'EGA.

CONSIGLIO

Potete aumentare gli intervalli di sovrascansione orizzontale e verticale dell'EGA nelle modalità a 350 linee modificando i parametri di temporizzazione orizzontale e verticale del CRTC. Un bordo ragionevole, largo almeno quanto quello visualizzato dalla VGA, può essere ottenuto aggiungendo uno o due caratteri al valore del totale orizzontale e otto o dieci linee di scansione al valore di totale verticale. I corrispondenti valori di temporizzazione dei registri di soppressione e di ritracciamento verticale ed orizzontale devono essere modificati di conseguenza (vedere Figura 37).
Il problema della riprogrammazione del CRTC in questo modo è che le frequenze orizzontale e verticale che controllano il monitor risultano in qualche modo inferiori rispetto ai valori nominali. Ad esempio, con i valori di CRTC mostrati nella Figura 37, la velocità di scansione orizzontale diventa 16,257 MHz : (94 caratteri/linea x 8/carattere), cioè 21,62 KHz, che equivale all'uno percento in meno della frequenza nominale di scansione orizzontale di 21,85 KHz. Analogamente, la velocità di scansione verticale diventa 21,62 KHz : 374 linee, cioè 58 Hz, quasi il 4 percento in meno della normale frequenza di quadro di 60 Hz. Tuttavia, queste velocità di scansione sono di solito entro le tolleranze di un monitor compatibile EGA.

Come evitare l'effetto neve del CGA

Su CGA, le modalità di visualizzazione alfanumeriche presentano un particolare problema di programmazione ogni qualvolta si affronta la velocità dell'output su schermo. Nelle modalità alfanumeriche dovete programmare in modo accurato per prevenire le interferenze con la visualizzazione quando leggete o scrivete i dati nel buffer del video del CGA.

Modalità alfanumeriche a 16 colori 80 per 25:

Registro CRTC	Funzione	Impostazione (default)
0	Totale orizzontale	5CH (5BH)
2	Inizio soppressione orizzontale	54H (53H)
3	Fine soppressione orizzontale	3CH (37H)
4	Inizio ritracc. orizzontale	52H (51H)
5	Fine ritracciamento orizzontale	5CH (5BH)

(continua)

Registro CRTC	Funzione	Impostazione (default)
6	Totale verticale	76H (6CH)
10H	Inizio ritracciamento verticale	64H (5EH)
11H	Fine ritracciamento verticale	25H (2BH)
15H	Inizio soppressione verticale	64H (5EH)
16H	Fine soppressione verticale	11H (0AH)

Modalità grafiche a 16 colori 640 per 350:

Registro CRTC	Funzione	Impostazione (default)
0	Totale orizzontale	5CH (5BH)
2	Inizio soppressione orizzontale	53H (53H)
3	Fine soppressione orizzontale	3CH (37H)
4	Inizio ritracc. orizzontale	53H (52H)
5	Fine ritracciamento orizzontale	00H (00H)
6	Totale verticale	76H (6CH)
10H	Inizio ritracciamento verticale	64H (5EH)
11H	Fine ritracciamento verticale	25H (2BH)
15H	Inizio soppressione verticale	64H (5EH)
16H	Fine soppressione verticale	11H (0AH)

Figura 37. *Parametri del CRTC per aumentare la larghezza del bordo nelle modalità di visualizzazione EGA a 350 linee (i valori di default dei registri sono indicati tra parentesi).*

L'accesso diretto al contenuto del buffer del video del CGA dal programma presenta i propri pro e contro. Dal lato dei vantaggi, permette al vostro programma di controllare completamente il contenuto del buffer e quindi di quanto viene visualizzato. Il lato negativo è rappresentato dal fatto che quando la CPU e la circuiteria di ripristino schermo accedono contemporaneamente al buffer, delle interferenze, o "effetto neve", possono apparire sullo schermo. L'effetto neve potrebbe essere appena visibile o estremamente fastidioso a seconda della quantità di dati trasferiti al/dal buffer del video.

In generale, per evitare l'effetto neve dovete limitare gli accessi della CPU al buffer del video in quei momenti in cui i dati non vengono prelevati dal buffer per ripristinare lo schermo. In pratica, ciò significa che il vostro programma deve trasferire i dati al/dal buffer del video solo quando il raggio elettronico del monitor si trova in sovrascansione o in un intervallo di ritracciamento.

Questa sincronizzazione può essere raggiunta con diversi metodi, ma, sfortunatamente, tutti i metodi introducono una certa quantità di dipendenza dall'hardware nel vostro programma. Come regola generale, maggiore è la dipendenza dall'hardware e più velocemente il programma girerà su un CGA ma minori saranno le probabilità che funzionerà su un altro adattatore video.

Cancellazione dello schermo

Una tecnica per impedire le interferenze dello schermo su CGA è rappresentata dalla disattivazione del raggio elettronico ogni volta che si accede al buffer del video. A quel punto si lascia disattivo il raggio mentre i dati vengono trasferiti al/dal buffer del video. Questo metodo viene utilizzato nelle routine del BIOS ROM che effettuano lo scorrimento dello schermo.

Il momento migliore per cancellare lo schermo è quando sarebbe vuoto comunque, cioè all'inizio dell'intervallo di soppressione verticale. Se non disattivate il raggio elettronico durante l'intervallo di soppressione verticale, cancellerete lo schermo mentre viene ripristinato. Questo produrrebbe un fastidioso sfarfallio o delle strisce di interferenza.

La tecnica è semplice (vedere Listato 19). Il punto è quello di sincronizzare l'accesso al buffer con l'inizio di un intervallo di soppressione verticale. Per fare ciò, rilevate un intervallo quando non sta avvenendo la soppressione verticale. Aspettate quindi l'inizio del successivo intervallo di soppressione.

```
                TITLE   'Listato 19'
                NAME    DisplayText
                PAGE    55,132

;
; Nome:         DisplayText
;
; Funzione:     Visualizza una stringa alfanumerica senza interferenze su CGA
;
; Chiamante:    Microsoft C:
;
;                       int DisplayText1(buf,n,offset);
;
;                       char *buf;      /* buffer contenente testo in formato
;                                          alfanumerico CGA (alterna codici
;                                          carattere e byte attributo) */
;
;                       int n;          /* lunghezza buffer in byte */
;
;                       offset int senza segno     ; /* offset nel buffer
                                                          del video */
;

Set80X25        EQU     (1 SHL 0)       ; maschere di bit per reg. controllo
                                        ;  modalità
Set320X200      EQU     (1 SHL 1)
lackAndWhite    EQU     (1 SHL 2)
EnableVideo     EQU     (1 SHL 3)
Set640X200      EQU     (1 SHL 4)
EnableBlink     EQU     (1 SHL 5)

ARGbuf          EQU     word ptr [bp+4] ; indirizzamento parametri in
                                        ;  stackframe
ARGn            EQU     word ptr [bp+6]
```

```
ARGoffset       EQU     word ptr [bp+8]
TIMEOUT         EQU     6               ; limite loop di tempo limite
                                        ;  ritracc. oriz.


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC _DisplayText
_DisplayText    PROC    near

                push    bp              ; solita premessa C per accedere ai
                mov     bp,sp           ; parametri in stack frame e salvare i
                                        ;  registri
                push    si
                push    di

                mov     ax,0B800h
                mov     es,ax
                mov     di,ARGoffset    ; ES:DI -> destinazione nel buffer
                                        ;  del video
                mov     si,ARGbuf       ; DS:SI -> buffer sorgente
                mov     bx,ARGn
                shr     bx,1            ; BX := lunghezza buffer in parole
                mov     dx,3DAh         ; DX := Porta di stato del CGA
; attende inizio del periodo di soppressione verticale

L01:            mov     cx,TIMEOUT      ; CX := contatore di loop (valore
                                        ;  tempo limite)

L02:            in      al,dx           ; AL := stato del video
                test    al,8
                jnz     L02             ; loop se attivo ritracc. verticale
                test    al,1
                jz      L02             ; loop se non attivo ritrac. orizzon.

                cli                     ; disabilita interrupt durante
                                        ;  ritrac. oriz.

L03:            in      al,dx
                test    al,1
                loopnz  L03             ; loop fino a conclusione ritrac. o
                                        ;  tempo limite

                sti                     ; riabilita interrupt

                jz      L01             ; loop se non è scaduto tempo limite

; cancella lo schermo

                mov     dl,0D8h         ; DX := 3D8h (reg. controllo modal.)
                mov     al,(Set80X25 OR EnableBlink)
                out     dx,al           ; disattiva video

; copia i dati nel buffer del video

                mov     cx,bx           ; CX := lunghezza buffer in parole
                rep     movsw
```

```
; riabilita lo schermo

                or      al,EnableVideo
                out     dx,al

                pop     di              ; solito epilogo C per ripristinare
                pop     si              ;  reg. ed eliminare stack frame
                mov     sp,bp
                pop     bp
                ret

_DisplayText    ENDP

_TEXT           ENDS

                END
```

Listato 19. *Visualizzazione di testo alfanumerico su CGA cancellando lo schermo.*

La procedura per il rilevamento dell'inizio di un intervallo di soppressione verticale richiede di determinare per prima cosa un valore di tempo limite per l'intervallo di ritracciamento orizzontale (vedere Listato 7). Questo valore viene quindi utilizzato per attendere l'ultima scansione orizzontale nel bordo corrente. Quando l'ultimo intervallo di soppressione orizzontale raggiunge il tempo limite, inizia l'intervallo di soppressione verticale.

A questo punto, il vostro programma dovrebbe disabilitare esplicitamente il raggio elettronico azzerando il bit 3 del registro di controllo modalità del CGA (porta 3D8H). Quando questo bit è stato azzerato, il raggio elettronico viene disabilitato e lo schermo resta scuro. Mentre lo schermo è scuro, potete spostare i dati al/dal buffer del video senza provocare l'effetto neve. Quando il trasferimento dei dati è stato completato, ripristinate lo schermo impostando a 1 il bit 3 del registro di controllo modalità.

Non è necessario attendere un altro intervallo di soppressione verticale prima di riabilitare il raggio elettronico. Se il periodo durante il quale avete effettuato il trasferimento dei dati ha lasciato scuro lo schermo abbastanza a lungo da causare uno sfarfallio avvertibile, l'attesa del ritracciamento verticale successivo prolungherà solamente la durata dello sfarfallio. Se riabilitate lo schermo durante un ciclo di ripristino, lo sfarfallio peggiorerà nella parte superiore dello schermo, ma migliorerà nella parte inferiore. Nessuna delle due situazioni rappresenta l'ideale; sta a voi decidere quale alternativa è preferibile.

La quantità di tempo necessaria per accedere al buffer del video determina l'intervallo di tempo durante il quale il programma deve tenere scuro lo schermo. Ovviamente, più a lungo lo schermo resta scuro, più si percepirà lo sfarfallio. Se il vostro programma viene eseguito su uno dei membri più lenti della famiglia di PC IBM (PC o PC/XT), lo sfarfallio può diventare molto fastidioso.

Considerate cosa potrebbe accadere quando effettuerete lo scorrimento di una linea di un intero schermo di 80 per 25. All'interno del buffer del video devono essere spostati

4000 byte di dati. Su un PC IBM d'annata, con l'8088 a 4,77 MHz di cui è dotato, questo trasferimento di dati richiede circa 21 millisecondi. Dal momento che ogni quadro dura circa 16,7 millisecondi (1/60 di secondo), lo schermo resta scuro per circa 1 quadro e 1/3. Lo sfarfallio risultante risulta notevole, in particolare se il colore di sfondo non è nero. Su un PC dotato di una CPU più rapida, invece, il trasferimento dei dati richiede meno tempo, quindi lo sfarfallio risulta meno evidente.

Uso dell'intervallo di soppressione verticale

Una tecnica che evita il problema dello sfarfallio è costituita dall'accesso al buffer del video solo per la durata dell'intervallo di soppressione verticale. Tuttavia, questa tecnica rallenta il trasferimento dei dati, in quanto potete spostare solo un numero limitato di byte di dati durante un singolo intervallo di soppressione verticale.

I limiti in questo caso sono rappresentati dalla durata dell'intervallo di soppressione verticale (circa 4 millisecondi) e dalla velocità alla quale la CPU può spostare i dati nel buffer del video. Un 8088 da 4,77 MHz di un PC o di un PC/XT può spostare circa 450 word (900 byte) di dati prima del termine dell'intervallo di soppressione verticale e l'effetto neve risulta inevitabilmente visibile. Ovviamente, un PC con una maggiore velocità di clock o con un 80286 o un 80386 può spostare più dati durante un singolo intervallo di soppressione verticale.

Uso dell'intervallo di soppressione orizzontale

Se la vostra routine di output su video si sincronizza con l'inizio degli intervalli di soppressione orizzontale, avete circa 7 microsecondi per accedere al buffer del video alla fine di ogni linea di scansione senza provocare l'effetto neve (vedere il Listato 20). Sebbene 7 microsecondi possano sembrare pochissimo tempo, sono sufficienti per spostare 2 byte al/dal buffer del video senza provocare interferenze. Dal momento che ogni quadro contiene 200 intervalli di soppressione orizzontale, potete aumentare significativamente le prestazioni sfruttandole tutte.

```
                TITLE   'Listato 20'
                NAME    DisplayText
                PAGE    55,132

;
; Nome:         DisplayText
;
; Funzione:     Visualizza una stringa alfanumerica senza interferenze su CGA
;
; Chiamante:    Microsoft C:
;                       int DisplayText(buf,n,offset);
;
```

```
;                       char *buf;            /* buffer contenente testo in
;                                                formato alfanumerico CGA
;                                                (alterna codici carattere
;                                                e byte di attributo) */
;
;                       int n;                /* lunghezza buffer in byte*/
;
;                       offset int senza segno ;    /* offset nel buffer
                                                       del video */
;

ARGbuf          EQU     word ptr [bp+4]
ARGn            EQU     word ptr [bp+6]
ARGoffset       EQU     word ptr [bp+8]
TIMEOUT         EQU     6                     ; limite loop di tempo limite
                                              ;  orizzontale
VBcount         EQU     250                   ; numero di word da
                                              ;  scrivere durante
                                              ;  intervallo di soppressione
                                              ;  verticale

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC _DisplayText
_DisplayText    PROC    near

                push    bp                    ; solita premessa C per
                                              ;  accedere ai parametri in
                                              ;  stack frame e salvare i
                mov     bp,sp                 ;  i registri
                push    si
                push    di

                mov     ax,0B800h
                mov     es,ax
                mov     di,ARGoffset          ; ES:DI - destinazione nel
                                              ;  buffer del video
                mov     si,ARGbuf             ; DS:SI - buffer sorgente
                mov     cx,ARGn
                shr     cx,1                  ; CX := lunghezza buffer in
                                              ;  parole

                mov     dx,3DAh               ; DX := Porta di stato CGA

; scrive durante restante intervallo di soppressione verticale

L01:            mov     bx,cx                 ; salva lunghezza buffer
                                              ;  in BX
                mov     cx,TIMEOUT            ; CX := tempo limite
                                              ;  orizzontale
                cli                           ; disabilita interrupt
                                              ;  durante loop

L02:            in      al,dx                 ; AL := stato video
                test    al,1
                loopnz  L02                   ; loop mentre 'abilita
                                              ;  visualizzazione' è
```

```
                                        ;  inattivo
            jz      L03                 ; salta se loop non ha
                                        ;  superato tempo limite

            movsw                       ; copia una word
            sti
            mov     cx,bx               ; CX := lunghezza buffer
            loop    L01
            jmp     short L10           ; uscita (intera stringa
                                        ;  copiata)
; scrive durante intervalli di soppressione orizzontale

L03:        sti
            mov     cx,bx               ; ripristina CX

L04:        lodsw                       ; AL := codice carattere
                                        ; AH := attributo
            mov     bx,ax               ; BX := carattere e attributo

            push    cx                  ; salva contatore loop
                                        ;  word
            mov     cx,TIMEOUT          ; CX := limite loop tempo
                                        ;  limite

            cli                         ; disabilita interrupt durante
                                        ;  una linea di scansione
L05:        in      al,dx
            test    al,1
            loopnz  L05                 ; loop durante soppressione
                                        ;  orizzontale
                                        ;  fino a scadere tempo
                                        ;  limite
            jnz     L07                 ; salta se supera tempo
                                        ;  limite (è iniziata
                                        ;  soppressione verticale)
L06:        in      al,dx
            test    al,1
            jz      L06                 ; loop mentre è attivo
                                        ;  'abilita visualizzazione'

            xchg    ax,bx               ; AX := carattere & attributo
            stosw                       ; copia 2 byte nel buffer del
                                        ;  video

            sti                         ; riabilita interrupt
            pop     cx                  ; CX := contatore loop word
            loop    L04

            jmp     short L10           ; uscita (intera stringa
                                        ;  copiata)



; scrive durante un intero intervallo di soppressione verticale

L07:        pop     bx                  ; BX := contatore loop word
            dec     si
```

```
                dec     si                      ; DS:SI - word da copiare
                                                ;  dal buffer

                mov     cx,VBcount              ; CX := # di word da
                                                ;  copiare
                cmp     bx,cx
                jnb     L08                     ; salta se restano più di
                                                ;  VBcount word
                                                ;  nel buffer

mov             cx,bx                           ; CX := # di word restanti
                                                ;  nel buffer
                xor     bx,bx                   ; BX := 0
                jmp     short L09

L08:            sub     bx,cx                   ; BX := (# di word
                                                ;  restanti) - VBcount

L09:            rep     movsw                   ; copia nel buffer del video

                mov     cx,bx                   ; CX := # di word restanti
                test    cx,cx
                jnz     L01                     ; loop fino a visualizzazione
                                                ;  buffer terminata

L10:            pop     di                      ; solito epilogo C per
                                                ;  ripristinare i registri ed
                pop     si                      ;  eliminare stack frame
                mov     sp,bp

                pop     bp
                ret

_DisplayText    ENDP

_TEXT           ENDS

                END
```

Listato 20. *Visualizzazione di testo alfanumerico su CGA durante intervalli di soppressione verticale ed orizzontale.*

Dal momento che l'intervallo di soppressione orizzontale è così breve, la sincronizzazione risulta difficile. La tecnica è analoga a quella utilizzata per la sincronizzazione con l'intervallo di ritracciamento verticale. In questo caso, si determina lo stato del segnale di abilitazione visualizzazione verificando la condizione del bit 0 del registro di stato del CRT (3DAH). Quando questo bit ha il valore 1, il segnale di abilitazione visualizzazione è disattivo e l'intervallo di soppressione orizzontale è attivo. Se pensate di utilizzare gli intervalli di soppressione orizzontale, occorre tenere presenti due considerazioni. La prima è che potreste utilizzare anche gli intervalli di soppressione verticale, dal momento che ci sono. Secondariamente, dovreste usare le istruzioni *MOVS* o *STOS* per eseguire l'effettivo trasferimento dei dati. L'istruzione *MOV* mem/reg più lenta può richiedere più

tempo di quanto duri l'intervallo di soppressione orizzontale e quindi l'effetto neve non verrà eliminato.

Le routine del BIOS ROM IBM che scrivono sul buffer del video durante il ritracciamento orizzontale utilizzano la sequenza:

```
mov     ax,bx
stosw
```

per spostare un carattere e l'attributo nel buffer senza l'effetto neve. Ciononostante, se utilizzate le stesse due istruzioni in un programma basato su RAM, vedrete l'effetto neve su un CGA che opera su un PC da 4,77 MHz. La ragione di ciò è che, al punto in cui vengono eseguite queste istruzioni, la coda da 4 byte di pre-fetch istruzioni dell'8088 ha spazio per solo due ulteriori byte. Ciò significa che il codice operativo *STOSW* non può essere prelevato. L'8088 deve prelevare il codice operativo dalla memoria prima che possa essere eseguito.

Quell'ultimo accesso alla memoria per prelevare l'istruzione *STOSW* è la causa della differenza. Dal momento che gli accessi alla ROM sono più veloci degli accessi alla RAM, il prelevamento dell'istruzione viene effettuato più rapidamente dalla ROM, e quindi non risulta visibile l'effetto neve in quanto *STOSW* può essere eseguita prima che termini l'intervallo di soppressione orizzontale. La routine del Listato 20 aggira l'ostacolo utilizzando *XCHG AX,BX* (un codice operativo da 1 byte) al posto di *MOV AX,BX* (un codice operativo da 2 byte). Questo evita il prelevamento dell'istruzione supplementare, quindi il codice viene eseguito abbastanza velocemente da evitare le interferenze dello schermo.

Si noti che gli interrupt sono disabilitati nel loop che attende l'inizio dell'intervallo di soppressione orizzontale. Se si fosse verificato un interrupt tra l'istruzione *JNZ L06* e la successiva istruzione *XCHG AX,BX*, l'intervallo di soppressione orizzontale sarebbe terminato molto prima che il controllo fosse restituito dal gestore dell'interrupt. La disabilitazione degli interrupt durante il trasferimento di ogni word nel buffer del video evita questa possibile perdita di sincronizzazione.

CONSIGLIO

La routine del Listato 20 non rileva mai la fine dell'intervallo di soppressione verticale, né conta le 200 scansioni orizzontali di ogni ciclo di ripristino di schermo. Al contrario, il numero dei byte che possono essere trasferiti nel corso di ogni intervallo di soppressione verticale (*VBcount*) viene determinato empiricamente per una situazione negativa limite (ad esempio, per un PC IBM da 4,77 MHz).
La ragione principale di questa imprecisione a proposito del numero di byte da trasferire durante gli intervalli di soppressione verticale deriva dal fatto che gli interrupt possono verificarsi in qualsiasi punto di una routine di output su schermo eccetto che

dove vengono disabilitati esplicitamente. Ad esempio, gli interrupt del clock e gli interrupt di tastiera possono verificarsi in qualsiasi momento. Visto che non è possibile disabilitare semplicemente tutti gli interrupt per tutto il tempo, dovete progettare delle routine di output su schermo in modo che assorbano il tempo imprevedibile richiesto dai gestori di interrupt.

Il problema dell'effetto neve viene evitato nel progetto hardware di ogni altro sottosistema di visualizzazione dei PC e dei PS/2 IBM, compresi l'MDA, l'EGA, l'MCGA e la VGA (e anche il PCjr). Inoltre, molti produttori per conto terzi di adattatori compatibili CGA progettano il loro hardware in modo da eliminare il problema. Ciò significa che i loop di sincronizzazione ritracciamento potrebbero non essere necessari in molte applicazioni.

Se eseguite un programma sia su CGA (con effetto neve) sia su un compatibile CGA (senza effetto neve), il programma dovrebbe tentare di determinare il tipo di hardware su cui sta operando (vedere Appendice C). Se il programma sta girando su una macchina senza effetto neve, può saltare qualsiasi loop di sincronizzazione verticale ed orizzontale. La breve operazione aggiuntiva necessaria per rilevare la presenza di un CGA viene ripagata da prestazioni enormemente migliorate dei sottosistemi di visualizzazione che non hanno problemi di effetto neve.

Uso dell'intero buffer del video

Nelle modalità di visualizzazione alfanumeriche, il CGA, l'EGA, l'MCGA e la VGA hanno molta più RAM disponibile nei loro buffer di video di quanta ne sia richiesta per visualizzare uno schermo di testo. In altre parole, potete visualizzare in un dato momento solo una parte dei dati del buffer del video. In effetti ciò che vedete sullo schermo è una "finestra" sul buffer del video.

Ad esempio, nelle modalità alfanumeriche 80 per 25 caratteri, solo 4000 byte (80x25x2 byte per carattere) vengono visualizzati in un dato momento. Tuttavia, il CGA ha 16 KB di RAM del video, quindi potete effettivamente memorizzare nel buffer quattro schermi da 80 per 25 di dati. Potete quindi programmare il controller del CRT del CGA in modo che visualizzi nel buffer qualsiasi blocco di 2000 caratteri consecutivi (4000 byte).

Pagine di visualizzazione del CGA

Per programmare il CGA in modo che visualizzi diverse aree del buffer, si aggiornano due registri del controller del CRT. Quando richiamate il BIOS ROM per selezionare una modalità di visualizzazione, il BOS inizializza il CRTC in modo che visualizzi i pri-

mi 4000 byte del buffer del video. Ciò viene ottenuto memorizzando 0, l'offset del primo carattere da visualizzare, nei registri di indirizzo iniziale del CRTC (0CH e 0DH).

Potete visualizzare qualsiasi area a scelta del buffer del video del CGA memorizzando un offset di buffer del video in word (non in byte) nei registri di indirizzo iniziale del CRTC. Il byte più significativo dell'offset deve essere memorizzato nel registro 0CH, il byte meno significativo nel registro 0DH. Ad esempio, il caricamento dei registri di indirizzo iniziale con l'offset in word della seconda riga (50H) provoca l'inizio della visualizzazione in quel punto (vedere Listato 21).

Il caricamento dei registri di indirizzo iniziale risulta essere un'operazione molto più veloce del trasferimento dei caratteri nel buffer del video. Di conseguenza, potreste considerare il buffer del video da 16 KB come uno schermo "virtuale" di 102 righe delle quali è possibile visualizzarne solo 25 per volta. Quando il buffer del video è pieno di testo, potete visualizzare rapidamente qualsiasi blocco di 25 righe consecutive semplicemente cambiando il valore dei registri di indirizzo iniziale del CRTC.

```
mov     ax,40h
mov     es,ax       ; ES := segmento dati del BIOS del video
mov     dx,es[63h]  ; DX := ADDR_6845

mov     al,0Ch      ; AL := numero registro (indirizzo iniziale
                    ;  più significativo)

out     dx,al
inc     dx          ; DX := 3x5h
mov     al,HiByte   ; AL := byte più significativo dell'offset
                    ;  iniziale
out     dx,al
dec     dx          ; DX := 3x4h

mov     al,0Dh      ; AL :=numero registro (indirizzo iniziale
                    ;  meno significativo)
out     dx,al
inc     dx          ; DX := 3x5h
mov     al,LoByte   ; AL := byte meno significativo dell'offset
                    ;  iniziale
out     dx,al
mov     ah,HiByte   ; AX := inizio offset in parole

shl     ax,1        ; AX := offset in byte
mov     es:[4Eh],ax ; aggiorna CRT_START
```

Listato 21. *Impostazione dei registri di indirizzo iniziale del CRTC.*

Ogni volta che aggiornate i registri di indirizzo iniziale, dovete aggiornare anche la word dell'area di visualizzazione del BIOS a 0040:004E (CRT_START). Ciò aiuta a mantenere la funzionalità delle chiamate del BIOS video e dell'MS-DOS.

Invece di decidere voi quali aree del buffer del video devono essere visualizzate, potreste trovare più comodo adottare il modello concettuale del BIOS ROM, che supporta quattro "pagine" virtuali da 80 per 25 (o otto 40 per 25) nel buffer del video del CGA. Per semplificare l'indirizzamento, ogni pagina inizia ad indirizzi multipli di 1 KB (1024 byte). Le quattro pagine da 80 per 25 iniziano perciò a B800:0000, B800:1000, B800:2000 e B800:3000. Potete visualizzare selettivamente qualsiasi pagina video richiamando la funzione 05H di INT 10H (vedere Listato 22).

```
mov     al,Vpage    ; AL := numero di pagina video
mov     ah,5        ; AH := numero di funzione INT 10h
int     10h
```

Listato 22. *Selezione della pagina video utilizzando il BIOS ROM.*

CONSIGLIO

Una tecnica che può migliorare le prestazioni del CGA è quella di visualizzare una pagina video mentre ne riempite un'altra (non visualizzata) con i dati. Successivamente visualizzate la pagina appena riempita rendendo disponibile la pagina precedente per ulteriori trasferimenti di dati. Progettate la vostra interfaccia utente in modo che mentre l'utente legge lo schermo, una pagina non visualizzata viene riempita con lo schermo successivo di informazioni. L'uso attento delle pagine di visualizzazione può rendere gli aggiornamenti dello schermo "istantanei".
Dovete comunque evitare le interferenze dello schermo utilizzando una delle tecniche per la sincronizzazione dell'aggiornamento con gli intervalli di soppressione verticale e orizzontale, anche se scrivete su un'area non visualizzata del buffer.

Pagine video dell'EGA, dell'MCGA e della VGA

Con EGA, MCGA e VGA, le tecniche per l'uso della RAM di visualizzazione sono analoghe a quelle utilizzate su CGA. I registri di indirizzo iniziale del controller del CRT sono collegati agli stessi indirizzi di porta di I/O del CRTC del CGA. Inoltre, il BIOS del video supporta le pagine video con la stessa interfaccia impiegata per il CGA. Ciò semplifica la scrittura dei programmi che devono essere eseguiti su tutti questi adattatori video.

CONSIGLIO

Una funzione comoda del CRTC dell'EGA, dell'MCGA, della VGA e di alcuni adattatori simili al CGA è la possibilità di leggere, oltre che scrivere, i registri di indirizzo iniziale. Questa

funzione può risultare utile nella programmazione diretta di questi registri, in quanto potete determinare il loro contenuto in un qualsiasi momento semplicemente ispezionandoli.

Controllo del cursore

Il controller del CRT controlla anche la dimensione e la locazione sullo schermo del cursore hardware nelle modalità alfanumeriche. Si specifica la dimensione del cursore caricando un registro del CRTC con i valori che indicano le linee superiore ed inferiore. La linea superiore è 0; il valore per la linea inferiore dipende dalla dimensione della matrice di carattere visualizzata (7 per una matrice 8 per 8 e 0DH per una matrice 9 per 14). La locazione del cursore viene specificata con un offset in word nel buffer del video, esattamente come si specifica l'indirizzo iniziale del controller del CRT.

Dimensione del cursore su MDA e CGA

I registri 0AH e 0BH del CRTC controllano la dimensione del cursore su tutti gli adattatori video dei PC e dei PS/2 IBM. Su MDA e su CGA, i cinque bit meno significativi del registro 0AH (inizio cursore) indicano la linea superiore del cursore visualizzato. I cinque bit meno significativi del registro 0BH (fine cursore) specificano la linea inferiore.

La modifica della dimensione del cursore hardware viene realizzata programmando questi due registri. Ad esempio, per visualizzare un cursore a "blocco", cioè un rettangolo che riempie uno spazio completo di carattere, impostate il registro di inizio cursore a 0 e il registro di fine cursore a un valore pari all'altezza della matrice del carattere meno uno. Per visualizzare il cursore di default del BIOS ROM, impostate i registri di inizio cursore e di fine cursore sui valori per le ultime due linee della matrice di carattere, come viene mostrato nel Listato 23.

Nella maggior parte delle applicazioni, tuttavia, potete utilizzare la funzione 1 di INT 10H (imposta tipo cursore) per modificare la dimensione del cursore. L'uso di questa funzione assicura la compatibilità con il BIOS del video su tutti i sottosistemi di visualizzazione dei PC e dei PS/2 IBM. Sebbene l'esecuzione di un interrupt software e della routine del BIOS risultino più lente della programmazione diretta del CRTC, in generale la dimensione del cursore viene modificata così raramente da rendere insignificante il leggero rallentamento del programma.

Inoltre, la routine del BIOS mantiene la dimensione corrente del cursore in due byte nell'area dei dati di visualizzazione a 0040:0060 (CURSOR_MODE). Su MDA e su CGA, i registri di inizio cursore e di fine cursore del CRTC sono registri di sola lettura, quindi potreste anche utilizzare il BIOS per conoscere lo stato corrente del cursore.
Il byte a 0040:0060 rappresenta il valore del registro 0AH del 6845 (inizio cursore), ed il

byte successivo, a 0040:0061, rappresenta il registro 0BH (fine cursore). Se aggirate la routine del BIOS e programmate direttamente il 6845, tenete aggiornati i valori di CURSOR_MODE.

```
; aggiornamento diretto dei registri del CRTC
            mov     ax,40h
            mov     es,ax           ; ES := segmento dati del BIOS del
                                    ;  video
            mov     dx,es[63h]      ; DX := ADDR_6845

            mov     al,0Ah          ; AL := numero registro (inizio cursore)
            out     dx,al
            inc     dx              ; DX := 3x5h
            mov     al,TopLine      ; AL := linea di scansione superiore del
                                    ;  cursore
            out     dx,al
            dec     dx              ; DX := 3x4h

            mov     al,0Bh          ; AL :=numero registro (fine cursore)
            out     dx,al
            inc     dx              ; DX := 3x5h
            mov     al,BottomLine   ; AL := linea di scansione inferiore del
                                    ;  cursore
            out     dx,al

            mov     ah,TopLine      ; AX := linee superiore ed inferiore
            mov     es:[60h],ax     ; aggiorna CURSOR_MODE

; uso dell'interfaccia del BIOS del video

            mov     ch,TopLine
            mov     cl,BottomLine
            mov     ah,1            ; AH := numero funzione INT 10h
            int     10h
```

Listato 23. *Impostazione della dimensione del cursore.*

Locazione del cursore su MDA e CGA

Per controllare la posizione del cursore, caricate l'offset di buffer nella parte alta (0EH) e bassa (0FH) del registro di posizione cursore del CRTC (vedere Listato 24). L'offset della posizione del cursore è relativo all'inizio del buffer del video. Se avete modificato i registri di indirizzo iniziale del CRTC, dovete regolare il nuovo offset di indirizzo iniziale nel calcolare l'offset della posizione del cursore.

```
; aggiornamento diretto dei registri del CRTC
            mov     ax,40h
            mov     es,ax       ; ES := segmento dati del BIOS del video
            mov     dx,es[63h]  ; DX := ADDR_6845
```

```
            mov     al,0Eh      ; AL := numero registro (posizione cursore
                                  parte alta)
            out     dx,al

            inc     dx          ; DX := 3x5h
            mov     al,HiByte   ; AL := byte più significativo dell'offset
                                  del cursore
            out     dx,al
            dec     dx          ; DX := 3x4h

            mov     al,0Fh      ; AL :=numero registro (posizione cursore
                                  parte bassa)
            out     dx,al
            inc     dx          ; DX := 3x5h
            mov     al,LoByte   ; AL := byte meno significativo dell'offset
                                  del cursore
            out     dx,al

; uso dell'interfaccia del BIOS del video

            mov     dh,CursorRow
            mov     dl,CursorColumn
            mov     bh,VideoPage
            mov     ah,2        ; AH := numero funzione INT 10h
            int     10h
```

Listato 24. *Impostazione della posizione del cursore.*

Controllo del cursore su MCGA

Il CRTC dell'MCGA raddoppia i valori da voi memorizzati nei registri di inizio cursore e di fine cursore e raddoppia il numero di linee di scansione nel cursore visualizzato. Di conseguenza, la dimensione del cursore alfanumerico dell'MCGA è un multiplo di due linee di scansione.

Questo raddoppio dei valori di inizio cursore e di fine cursore permette di specificare dimensioni di default del cursore alfanumerico con gli stessi valori che avreste utilizzato su CGA. Ad esempio, nelle modalità alfanumeriche di default dell'MCGA, la matrice di carattere è alta 16 linee. Se impostate inizio cursore a 6 e fine cursore a 7, come fareste in una modalità alfanumerica CGA, vedrete il cursore MCGA in fondo alla matrice di carattere nelle linee da 0CH a 0FH. In questo modo i registri di inizio cursore e di fine cursore dell'MCGA emulano quelli del CGA a prescindere dalla matrice di carattere più alta dell'MCGA.

Tuttavia, esistono diverse differenze nel modo in cui l'MCGA interpreta i valori di inizio cursore e di fine cursore (vedere Figura 38). Su MCGA, solo i quattro bit meno significativi dei valori di inizio cursore e di fine cursore sono significativi. Inoltre, dal momento che la matrice di carattere può essere alta al massimo 16 linee di scansione, i valori di inizio cursore e di fine cursore sono di solito limitati alla gamma da 0 a 7. I va-

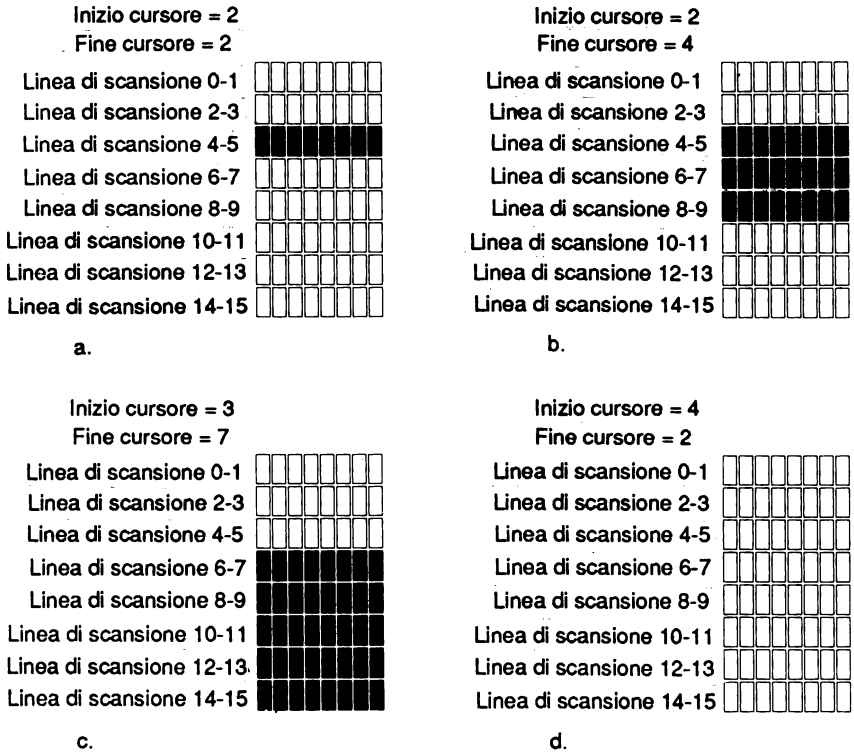
lori maggiori di 7 possono produrre un cursore che prosegue nella parte superiore della matrice di carattere (vedere Figura 38e).

Controllo del cursore su EGA e VGA

Su EGA e su VGA, i registri di inizio cursore, fine cursore, parte alta e bassa della posizione cursore sono locati agli stessi numeri di registro del CRTC dell'MDA e del CGA. Ciò potrebbe far sorgere dei problemi se siete interessati alla trasportabilità e se avete bisogno di scrivere direttamente sui registri del CRTC. Questo perché i registri di inizio cursore e di fine cursore dell'EGA e della VGA non funzionano esattamente come quelli su MDA, CGA o MCGA.

Su EGA, il valore da voi specificato per il registro di fine cursore deve essere maggiore di 1 rispetto alla linea inferiore del cursore (vedere Figura 39). Il controller del CRT dell'EGA visualizza il cursore alfanumerico a partire dalla linea di scansione carattere specificata nel registro di inizio cursore fino alla linea specificata nel registro fine cursore meno 1.

Se il valore di fine cursore è inferiore al valore di inizio cursore, il cursore prosegue nella parte superiore della matrice di carattere. Se i quattro bit meno significativi dei valori di inizio cursore e di fine cursore sono identici, il cursore apparirà solo sulla singola linea specificata nel registro di inizio cursore. Infine, il valore di fine cursore deve essere inferiore al numero di linee di scansione della matrice di carattere. In caso contrario, il controller del CRT visualizza un cursore a piena altezza a prescindere dal valore del registro di inizio cursore.



(continued)

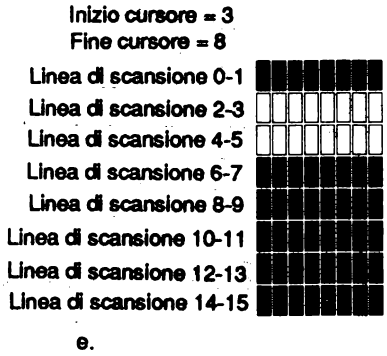
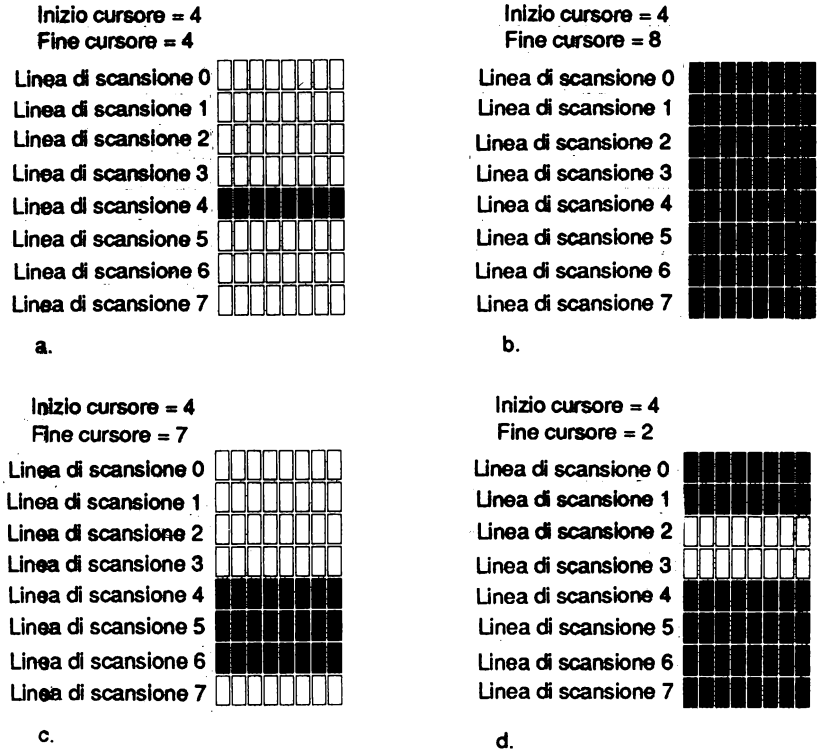


Figura 38. *Esempi di cursore alfanumerico MCGA per una matrice di carattere 8 per 16.*

I registri di inizio cursore e di fine cursore della VGA (vedere Figura 40) funzionano in un modo leggermente diverso da come funzionano su EGA. Il valore di fine cursore della VGA indica l'ultima linea del cursore visualizzato (non l'ultima linea più 1), e il cursore visualizzato non prosegue sulla parte superiore della matrice di carattere se il valore di fine cursore è inferiore al valore di inizio cursore (vedere Figure 39 e 40).



(continua)

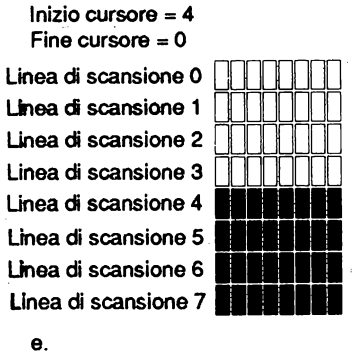


Figura 39. *Esempi di cursore alfanumerico EGA per una matrice di carattere 8 per 8.*

CONSIGLIO

I bit 5 e 6 del registro di fine cursore (0BH) su EGA e su VGA controllano lo spostamento verso destra del cursore. Se i bit 5 e 6 non sono a 0, il cursore appare uno, due o tre caratteri a destra della locazione specificata dai registri di locazione cursore. Per la maggior parte delle applicazioni, lo spostamento del cursore dovrebbe essere 0.

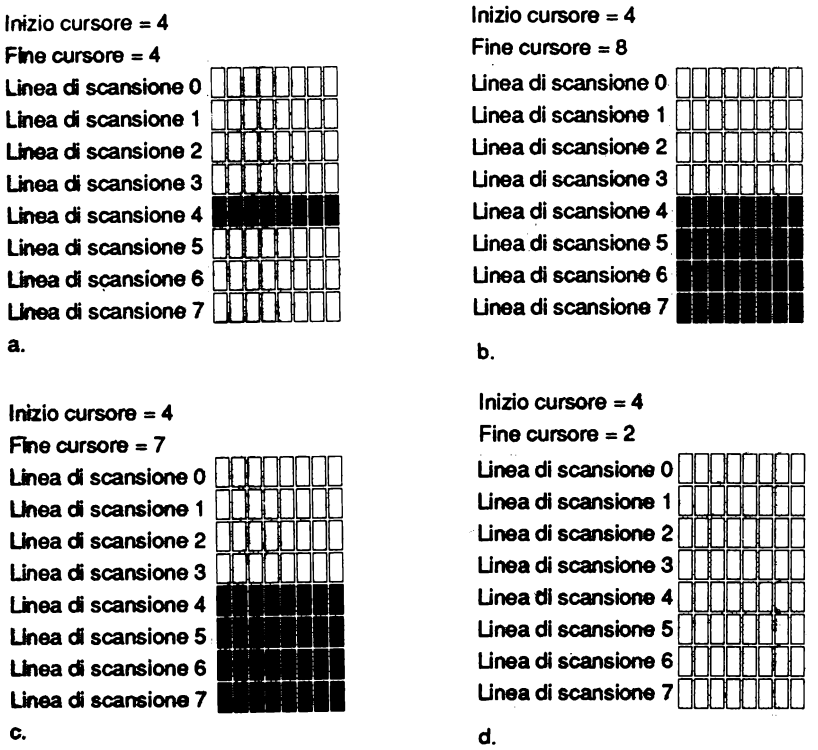


Figura 40. *Esempi di cursore alfanumerico VGA per una matrice di carattere 8 per 8.*

Emulazione del cursore del BIOS ROM

La routine del BIOS ROM per la funzione 01H di INT 10H usa i valori dei registri CH e CL dell'80x86 per programmare i registri di inizio cursore e di fine cursore del CRTC (vedere Listato 23). Su MDA o su CGA, questi valori vengono semplicemente copiati nei registri del CRTC. Su EGA o su VGA, invece, il BIOS può scalare questi valori relativamente ad una matrice di carattere di 8 linee e programmare il CRTC con i risultati ottenuti. Questa scalatura viene definita "emulazione cursore" nei manuali tecnici dell'IBM.

Quando l'emulazione del cursore del BIOS ROM è attiva, i valori da voi specificati nella funzione 01H di INT 10H rappresentano la posizione dell'inizio e della fine del cursore visualizzato, relativamente ad una matrice di carattere da 8 linee. Quando la matrice di carattere effettiva è più estesa di 8 linee, la routine del BIOS modifica i valori di inizio cursore e di fine cursore per mantenere la locazione relativa del cursore nella matrice.

Considerate cosa avviene, ad esempio, quando chiamate la funzione 01H di INT 10H con CH=6 e CL=7. Se la matrice di carattere è alta 8 linee, il cursore appare sulle due linee inferiori (questo è il solito cursore delle modalità di visualizzazione a 200 linee). Se la matrice di carattere è alta 14 linee, però, la routine del BIOS modifica i valori di inizio cursore e di fine cursore in modo che il cursore appaia nella parte inferiore della matrice, cioè, sulle linee 0BH e 0CH. Di conseguenza, l'emulazione del cursore permette ai programmi che modificano la dimensione del cursore con la funzione 01H di INT 10H di funzionare senza alcuna modifica a prescindere dalla dimensione della matrice del carattere.

Il BIOS realizza l'emulazione del cursore nelle funzioni INT 10H ogni volta che il bit 0 del byte INFO dell'area dati di visualizzazione (0040:0087) è impostato a 0 (questo è il default all'accensione per EGA e per VGA). Potete disabilitare l'emulazione del cursore impostando a 1 questo bit prima di richiamare la funzione 01H di INT 10H. Su EGA, dovete impostare e azzerare direttamente il bit, ma su VGA, dovreste utilizzare la funzione 12H di INT 10H per impostare il valore del bit.

CONSIGLIO

Su EGA, l'emulazione del cursore viene realizzata aggiungendo 5 a qualsiasi valore di inizio cursore o di fine cursore maggiore di 4. Questo funziona bene quando la matrice di carattere è quella di default, alta 14 linee. Per le matrici di carattere di altezza diversa, tuttavia, questo semplice algoritmo calcola in modo errato i valori di inizio cursore e di fine cursore. Dovreste quindi disabilitare l'emulazione del cursore quando programmate il generatore di caratteri dell'EGA per modificare la dimensione della matrice di carattere (vedere Capitolo 10).
Su VGA, il calcolo per l'emulazione del cursore tiene conto dell'altezza della matrice di carattere, quindi il cursore emulato

viene visualizzato correttamente a prescindere dalle dimensioni della matrice di carattere.

Un cursore invisibile

Potete rendere "invisibile" il cursore programmando il controller del CRT in modo che lo visualizzi in una locazione fuori dallo schermo. Per fare ciò impostate i registri di cursore alto e di cursore basso su un offset di buffer non visualizzabile. Un altro metodo per rendere invisibile il cursore è quello di caricare i registri di inizio cursore e di fine cursore con valori inferiori alla matrice di carattere visualizzata. Su MDA, CGA e VGA, caricate il registro di inizio cursore con il valore 20H per fare scomparire il cursore. Su EGA, impostate inizio cursore con un valore maggiore o uguale al numero di linee della matrice di carattere ed impostate fine cursore a 0 (vedere Listato 25).

```
mov     cx,2000h    ; CH := linea di scansione superiore per il
                    ;  cursore
                    ; CL := linea di scansione inferiore per il
                    ;  cursore
mov     ah,1        ; AH := numero funzione INT 10h
int     10h
```

Listato 25. *Un cursore alfanumerico invisibile per gli adattatori video IBM.*

4
Modalità grafiche

Uso delle modalità grafiche

Mappatura dei pixel sullo schermo
CGA - HGC - EGA
Scheda InColor Hercules - MCGA e VGA

Coordinate dei pixel
Scalatura delle coordinate dei pixel
Deformazione

Attributi di visualizzazione dei pixel
CGA - HGC - EGA
Scheda InColor Hercules - MCGA - VGA

Questo capitolo tratta i punti fondamentali della programmazione delle modalità grafiche su CGA, EGA, MCGA, VGA e schede Hercules. Per prima cosa il capitolo descrive come i pixel vengono rappresentati nel buffer del video e come sono mappati sullo schermo. Successivamente vengono trattati gli attributi di visualizzazione dei pixel e cioè come determinare il colore, la luminosità e il lampeggiamento di un pixel.

Uso delle modalità grafiche

Nelle modalità grafiche, il programma può manipolare il colore di ogni pixel dello schermo. Per questo motivo, le modalità grafiche sono talvolta chiamate modalità ad "indirizzamento di tutti i punti" (APA). Dal momento che avete il controllo di ogni pixel dell'immagine visualizzata, potete costruire immagini geometriche complesse, riempire aree arbitrarie dello schermo con colori pieni o con miscele di colori e visualizzare immagini animate che si spostano senza scatti sullo schermo.

La maggior parte dei programmatori, tuttavia, utilizzano le modalità grafiche solo quando il controllo pixel per pixel dello schermo è essenziale per un'applicazione. Il motivo: il prezzo da pagare per il controllo totale dello schermo è un aumento considerevole della complessità del codice sorgente e una diminuzione nelle prestazioni. Un semplice confronto tra la quantità di dati richiesti per visualizzare un intero schermo di informazioni nelle modalità alfanumeriche e quelli richiesti nelle modalità grafiche permette di scoprire perché.

Ad esempio, per visualizzare 25 righe di testo a 80 colonne e a 16 colori in modalità alfanumerica su un EGA, è necessario memorizzare 4.000 byte (80x25x2) nel buffer video. Con un monitor da 350 linee, il testo viene visualizzato con una risoluzione di 640 per 350 pixel. Per ottenere la stessa risoluzione in una modalità grafica a 16 colori sono necessari 112.000 byte (640x350x4 bit per pixel : 8 bit per byte). Ovviamente, un programma che deve manipolare 112.000 byte di dati è molto più complesso e più lento di un programma che deve manipolare solo 4.000 byte.

Naturalmente, la perdita in prestazioni derivante dall'uso dell'output su video in modalità grafica è meno evidente se utilizzate un computer più veloce, come le macchine basate su 80286 o 80386: CPU che operano ad una velocità di clock superiore. Tuttavia, prima di addentrarvi nella programmazione delle modalità grafiche, dovreste considerare attentamente le alternative. Le modalità alfanumeriche sono sufficienti per visualizzare del testo e delle semplici immagini grafiche a blocchi e, quindi, per la maggior parte delle applicazioni normali.

CONSIGLIO

Un'alternativa in alcune applicazioni è quella di utilizzare un adattatore video che abbia un generatore di caratteri alfanumerici in grado di visualizzare set di caratteri memorizzati in RAM (l'EGA, l'MCGA, la VGA, l'HGC+ e la scheda InColor hanno tutti questa possibilità). Con questi adattatori, potete progettare "caratteri" che sono in effetti delle porzioni di immagini grafiche più grandi e quindi assemblare le porzioni per comporre un'immagine completa in una modalità alfanumerica (il Capitolo 10 spiega la tecnica in dettaglio).

Mappatura dei pixel sullo schermo

Gli adattatori grafici dei PC e dei PS/2 memorizzano i dati dei pixel come gruppi di bit che rappresentano i valori dei pixel. Il colore di ogni pixel sullo schermo viene determinato, direttamente o indirettamente, dal suo valore. Inoltre, nessun valore di pixel è mai rappresentato da più di otto bit, quindi in ogni byte del buffer del video sono rappresentati uno o più pixel.

Il formato della mappa dei pixel o della bitmap nel buffer del video dipende dal numero dei bit richiesti per rappresentare ogni pixel, oltre che dall'architettura della RAM del video. Ovviamente, il numero dei colori che una data modalità grafica può visualizzare contemporaneamente viene determinato dal numero di bit utilizzati per rappresentare ogni pixel.

Quando i valori dei pixel sono inferiori agli otto bit, i pixel vengono mappati in campi di bit da sinistra verso destra in ogni byte. Il pixel più a sinistra rappresentato in un dato byte si trova sempre nel bit (o nei bit) più significativo di quel byte. Ciò accade su tutti i sottosistemi di visualizzazione dei PC e dei PS/2.

Color Graphics Adapter

Su CGA, ogni pixel viene rappresentato da due bit, come nella modalità a 4 colori 320 per 200 (vedere Figura 41a) o da un bit, come nella modalità a 2 colori 640 per 200 (vedere Figura 41b). Dal momento che vengono utilizzati due bit per rappresentare i pixel nella modalità 320 per 200, un pixel può avere uno qualsiasi dei quattro valori di pixel, quindi questa modalità può visualizzare quattro colori diversi contemporaneamente. Solo un bit viene utilizzato per rappresentare i valori di pixel nella modalità 640 per 200, quindi questa modalità può visualizzare solo due colori per volta.

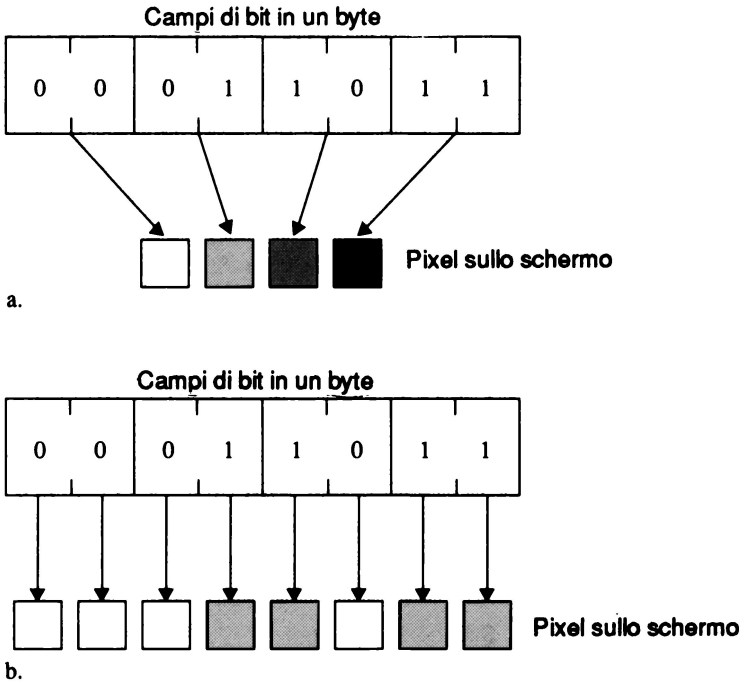


Figura 41. *Mappatura dei pixel nelle modalità grafiche CGA.*

I dati dei pixel sono mappati in due metà separate del buffer video da 16 KB del CGA. I dati per le 100 linee di scansione pari iniziano a B800:0000, mentre quelli per le linee di scansione dispari iniziano a B800:2000 (vedere Figura 42). Se le linee di scansione sono numerate consecutivamente a partire da 0, la metà del buffer del video nella quale viene rappresentata la linea di scansione *n*sima può essere determinata calcolando *n* MOD 2.

CONSIGLIO

Questa architettura a due pagine del buffer permette al controller del CRT del CGA di visualizzare 200 linee di dati grafici senza superare la capacità dei registri di temporizzazione verticale del CRTC da 7 bit. Nelle modalità grafiche CGA, il CRTC è impostato in modo da visualizzare 100 righe di "caratteri", ognuno dei quali alto due linee di scansione. La linea superiore (pari) di ogni carattere viene ricavata dalla prima metà del buffer del video e la linea inferiore (dispari) viene letta dalla seconda metà del buffer del video.

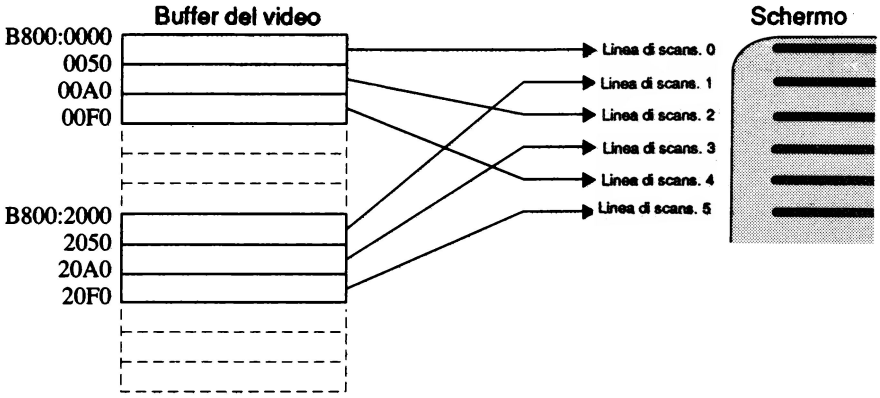


Figura 42. *Organizzazione del buffer del video nelle modalità grafiche CGA.*

Scheda grafica Hercules

Nella modalità grafica 720 per 348 su HGC e su HGC+, la rappresentazione dei pixel è analoga a quella della modalità grafica a 2 colori 640 per 200 del CGA. Un bit rappresenta ogni pixel, quindi sono disponibili solo due "colori" (pixel acceso e pixel spento).

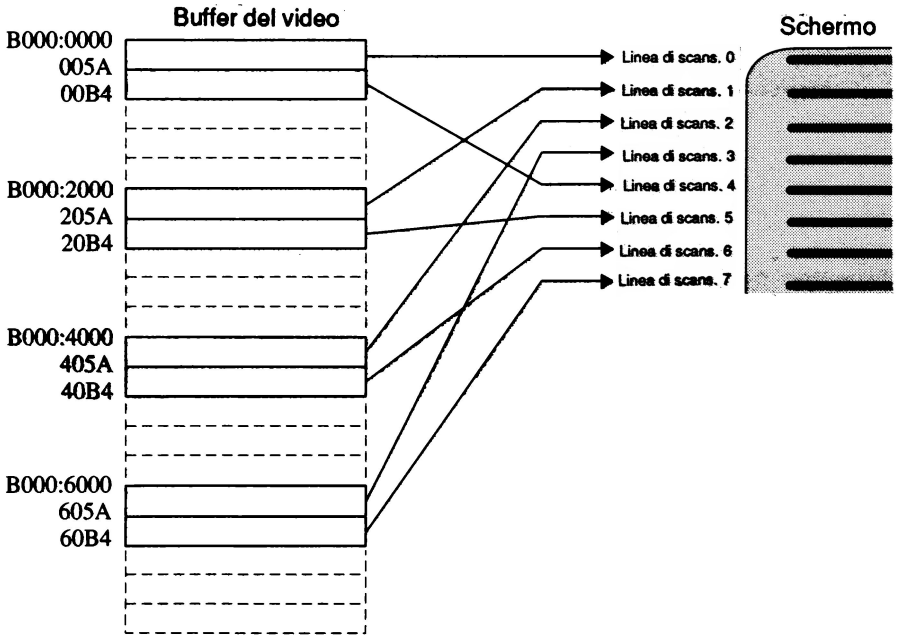


Figura 43. *Organizzazione del buffer del video nella modalità grafica Hercules.*

Le 348 linee da novanta byte di dati di pixel dell'HGC, però, sono memorizzate utilizzando quattro aree separate del buffer del video (vedere Figura 43), ognuna delle quali contiene ottantasette (348 : 4) linee di dati. Con questa organizzazione di buffer, l'area nel buffer nella quale viene rappresentata la *n*sima linea di scansione può essere determinata da *n* MOD 4.

Sugli adattatori video Hercules, la suddivisione in quattro pagine permette al CRTC di essere programmato in modo da visualizzare 87 righe di caratteri alti quattro linee di scansione (vedere Listato 9 nel Capitolo 2). Ognuna delle quattro linee di scansione di un "carattere" viene letta dalla locazione corrispondente in una delle quattro aree "separate" del buffer del video.

Enhanced Graphics Adapter

Quando l'EGA è configurato per emulare una modalità grafica CGA, i pixel sono mappati nel buffer del video esattamente come lo sarebbero su CGA. Tuttavia, nelle modalità grafiche originali dell'EGA (le modalità a 16 colori da 200 linee e tutte le modalità a 350 linee), i pixel sono sempre mappati a otto pixel per byte.

Questa mappatura è dettata dall'architettura del buffer del video dell'EGA. Il buffer del video da 256 KB è costituito da quattro mappe da 64 KB, o banchi paralleli di RAM. Le mappe sono parallele nel senso che occupano la stessa gamma di indirizzi nello spazio di indirizzo della CPU; il sequencer ed il controller grafico dell'EGA permettono di accedere alle mappe o individualmente o in parallelo (ulteriori dettagli su questo argomento si trovano nel Capitolo 5).

Il valore di un pixel viene determinato dai valori dei bit corrispondenti nello stesso offset di byte e nell'offset di bit di ogni mappa (vedere Figura 44). Per questo motivo, nelle modalità grafiche, le quattro mappe vengono chiamate piani di bit. Potreste immaginare il valore di ogni pixel come il risultato del concatenamento di un bit della stessa locazione di ogni piano di bit.

CONSIGLIO

La relazione delle mappe di memoria con i piani di bit viene modificata nelle modalità grafiche a 350 linee su un EGA IBM dotato di solo 64 KB di RAM di visualizzazione (per portare l'EGA originale IBM fino a 256 KB, dovete installare una piastra a connessione superiore, chiamata scheda di espansione della memoria grafica). Quando utilizzate la funzione 00H di INT 10H per selezionare le modalità grafiche 640 per 350 (la modalità 0FH o 10H) su un EGA con un buffer del video da 64 KB, la decodifica dell'indirizzo del buffer viene modificata in modo che gli indirizzi pari del buffer facciano riferimento alle mappe di numero pari e gli indirizzi dispari si riferiscano alle mappe di numero dispari (vedere la Figura 45).
In questo modo le quattro mappe del buffer video vengono con-

catenate, con le mappe 0 e 1 che compongono il piano di bit 0 e le mappe 2 e 3 che compongono il piano di bit 2. Le routine che accedono ai pixel del buffer del video devono conformarsi a questa relazione tra i piani di bit e gli indirizzi del buffer.

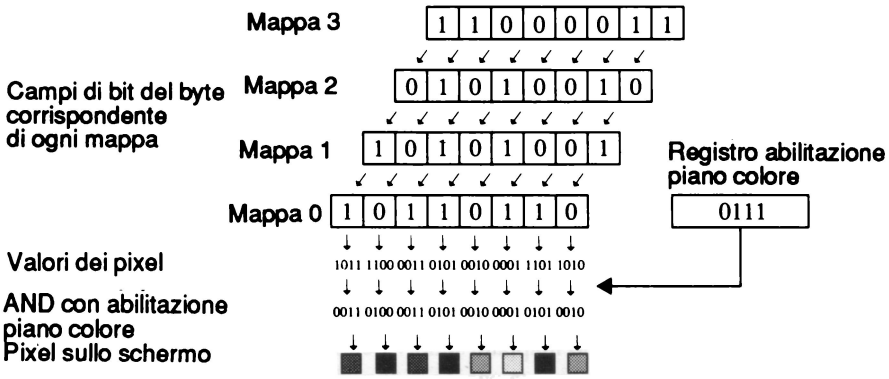


Figura 44. *Mappatura dei pixel nelle modalità grafiche originali dell'EGA.*

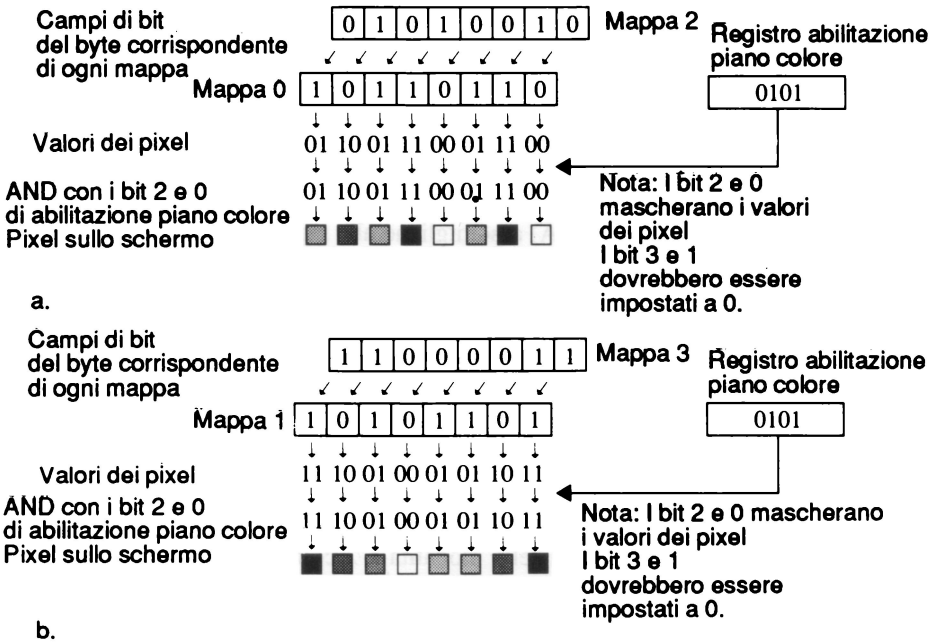


Figura 45. *Mappe del buffer del video nelle modalità grafiche a 350 linee (EGA con RAM video di 64 KB). I valori dei pixel agli indirizzi pari sono memorizzati nelle mappe 0 e 2 (Figura 45a); i pixel agli indirizzi dispari sono memorizzati nelle mappe 1 e 3 (Figura 45b).*

Nelle modalità grafiche EGA originali, non esiste la suddivisione linea per linea dei dati dei pixel nel buffer del video, come avviene nelle modalità grafiche del CGA e dell'HGC. Al contrario, le righe dei pixel sono mappate linearmente, come le righe di caratteri sono mappate linearmente nelle modalità alfanumeriche.

Scheda InColor Hercules

Nella sua modalità grafica 720 per 348, il buffer del video della scheda InColor ha quattro mappe parallele organizzate come quattro piani di bit paralleli. Come su EGA, il valore di un pixel viene determinato concatenando i bit corrispondenti di ogni piano di bit. Tuttavia, l'indirizzamento del buffer del video non è lineare, come lo è invece su EGA.

I pixel vengono memorizzati nel buffer del video della scheda InColor utilizzando la stessa architettura a quattro pagine impiegata dall'HGC e dall'HGC+. Nel buffer, 348 linee di 90 byte (720 pixel) vengono mappate suddividendole in quattro pagine a partire da B000:0000. Il buffer contiene anche due pagine di visualizzazione (come su HGC monocromatica), a B000:0000 e B000:8000. Questo aspetto del progetto della scheda InColor ne conserva la simmetria con le schede grafiche monocromatiche Hercules, ma la differenzia dall'EGA.

MCGA e VGA

Gli adattatori video dei PS/2 supportano tre modalità grafiche non presenti sui primi adattatori video dei PC. La modalità a 2 colori 640 per 480 (MCGA e VGA) e la modalità a 16 colori 640 per 480 (solo VGA) assomigliano alle modalità grafiche EGA originali: entrambe utilizzano una mappa di bit lineare a partire da A000:0000. Una simile mappa lineare di pixel viene anche utilizzata nella modalità a 256 colori 320 per 200 (MCGA e VGA), con un'importante differenza: ogni byte del buffer del video rappresenta un pixel (vedere Figura 46). Dal momento che esistono otto bit in un byte, ogni pixel può assumere uno qualsiasi dei 256 (2^8) diversi colori.

CONSIGLIO

Su VGA, la modalità a 2 colori 640 per 480 è quasi identica alla modalità a 16 colori 640 per 480. Tutti i quattro piani di bit restano attivi nella modalità a 2 colori nonostante sia sufficiente un piano di bit per memorizzare un intero schermo di pixel. L'unica differenza tra le due modalità è che il BIOS del video rende disponibili solo due colori di tavolozza nella modalità a 2 colori, mentre imposta 16 colori di tavolozza nella modalità a 16 colori.

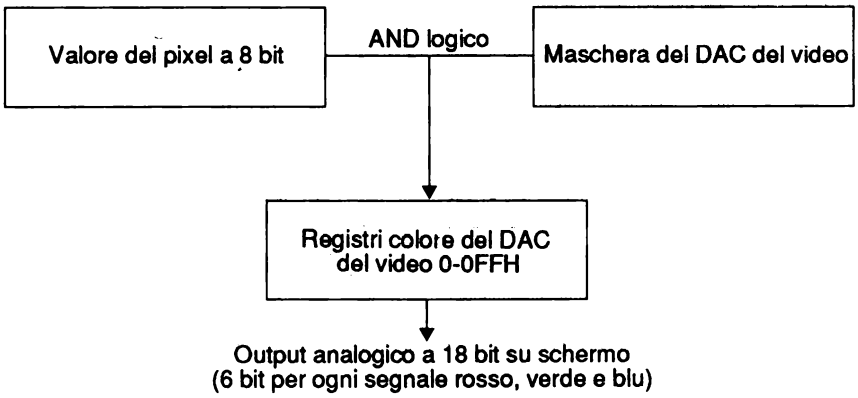


Figura 46. *Selezione del colore nella modalità a 256 colori 320 per 200 dell'MCGA e della VGA*

Coordinate dei pixel

Nelle modalità grafiche, il buffer del video può essere paragonato ad una matrice bidimensionale e piatta di pixel, con l' origine nell'angolo superiore sinistro. Ciò che si vede sullo schermo è un sottoinsieme dei pixel rappresentati nel buffer. Su CGA, il buffer del video può contenere solo uno schermo completo di pixel, quindi il primo byte del buffer rappresenta i pixel dell'angolo superiore sinistro dello schermo. Su EGA, MCGA e VGA invece, il buffer del video può memorizzare diversi schermi completi di pixel. Potete quindi selezionare quale area del buffer del video dovrà apparire sullo schermo.

Ogni pixel dello schermo può essere identificato da una coppia unica di coordinate (*x,y*) relative all'angolo superiore sinistro dello schermo. Ogni coppia (*x,y*) corrisponde anche ad un particolare offset di byte nel buffer del video e ad un offset di bit in quel byte Di conseguenza, avendo una data coordinata (*x,y*) di un pixel sullo schermo, potete calcolare la posizione all'interno del buffer del video nella quale il pixel è rappresentato.

La conversione da coordinate di pixel ai corrispondenti offset di byte e di bit è una delle operazioni più frequenti nella programmazione grafica dei video IBM. Gli esempi di programma dei Listati da 26 a 30 dimostrano come realizzare questa operazione in modo efficiente ed uniforme.

```
                TITLE   'Listato 26'
                NAME    PixelAddr04
                PAGE    55,132

;
; Nome:         PixelAddr04
;
; Funzione:     Determina indirizzo nel buffer del pixel in modalità
;               320x200 a 4 colori
;
; Chiamante:    AX = coordinata y (0-199)
;               BX = coordinata x (0-319)
;
; Ritorna:      AH = maschera di bit
;               BX = offset di byte nel buffer
;               CL = numero di bit da spostare verso sinistra
;               ES = segmento buffer del video
;


OriginOffset    EQU     0                       ; offset in byte di (0,0)
VideoBufferSegEQU       0B800h

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC PixelAddr04
PixelAddr04     PROC    near

                mov     cl,bl                   ; CL := byte meno significativo
                                                ;  di x

                xchg    ah,al                   ; AX := 100h * y
                shr     ax,1                    ; AL := 80h * (y&1)
                add     bh,al                   ; BX := x + 8000h*(y&1)
                xor     al,al                   ; AX := 100h*(y/2)
                add     bx,ax                   ; BX := x + 8000h*(y&1) +
                                                ;  100h*(y/2)
                shr     ax,1
                shr     ax,1                    ; AX := 40h*(y/2)
                add     bx,ax                   ; BX := x + 8000h*(y&1) +
                                                ;  140h*(y/2)
                shr     bx,1
                shr     bx,1                    ; BX := x/4 + 2000h*(y&1) +
                                                ;  50h*(y/2)
                add     bx,OriginOffset         ; BX := offset del byte nel
                                                ;  buffer video

                mov     ax,VideoBufferSeg
                mov     es,ax                   ; ES:BX := indirizzo del
                                                ;  byte del pixel

                mov     ah,3                    ; AH := maschera di bit non
                                                ;  spostata
                and     cl,ah                   ; CL := x & 3
```

```
                xor     cl,ah                   ; CL := 3 - (x & 3)
                shl     cl,1                    ; CL := # di bit da spostare
                                                ;  verso sinistra

                ret

PixelAddr04     ENDP

_TEXT           ENDS

                END
```

Listato 26. *Calcolo dell' indirizzo di un pixel nella modalità 320 per 200 a 4 colori.*

```
                TITLE   'Listato 27'
                NAME    PixelAddr06
                PAGE    55,132

;
; Nome:         PixelAddr06
;
; Funzione:     Determina indirizzo nel buffer del pixel nella modalità
;               640x200 a 2 colori
;
; Chiamante:    AX = coordinata y (0-199)
;               BX = coordinata x (0-639)
;
; Ritorna:      AH = maschera di bit
;               BX = offset del byte nel buffer
;               CL = numero di bit da spostare verso sinistra
;               ES = segmento del buffer del video
;


OriginOffset    EQU     0                       ; offset del byte di (0,0)
VideoBufferSegEQU       0B800h

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC PixelAddr06
PixelAddr06     PROC    near

                mov     cl,bl                   ; CL := byte meno significativo
                                                ;  di x

                xchg    ah,al                   ; AX := 100h * y
                shr     bx,1                    ; BX := x/2
                shr     ax,1                    ; AL := 80h*(y&1)
                add     bh,al                   ; BX := x/2 + 8000h*(y&1)
                xor     al,al                   ; AX := 100h*(y/2)
                add     bx,ax                   ; BX := x/2 + 8000h*(y&1) +
                                                ;  + 100h*(y/2)
```

```
                shr     ax,1
                shr     ax,1                    ; AX := 40h*(y/2)

                add     bx,ax                   ; BX := x/2 + 8000h*(y&1) +
                                                ;  140h*(y/2)
                shr     bx,1
                shr     bx,1                    ; BX := x/8 + 2000h*(y&1) +
                                                ;  50h*(y/2)
                add     bx,OriginOffset         ; BX := offset del byte nel
                                                ;  buffer del video

                mov     ax,VideoBufferSeg
                mov     es,ax                   ; ES:BX := indirizzo del byte
                                                ;  del pixel

                and     cl,7                    ; CL := x & 7
                xor     cl,7                    ; CL := numero di bit da
                                                ;  spostare a sinistra
                mov     ah,1                    ; AH := maschera di bit non
                                                ;  spostato

                ret

PixelAddr06     ENDP

_TEXT           ENDS

                END
```

Listato 27. *Calcolo dell'indirizzo di un pixel nella modalità 640 per 200 a 2 colori.*

La trasformazione delle coordinate di pixel in offset di buffer comporta una semplice logica. Iniziate calcolando l'offset dell'inizio della riga di pixel *y* (per le modalità grafiche CGA e Hercules, questo calcolo tiene conto dell'intercalazione del buffer del video). A questo valore, aggiungete l'offset del byte dell'*x*esimo pixel della riga. Infine, aggiungete l'offset del byte dell'inizio dell'area visualizzata del buffer del video per ottenere l'offset del byte definitivo del pixel.

```
PixelByteOffset = RowOffset (y) + ByteOffset (x) + OriginOffset
```

L'offset del bit del pixel all'interno del byte che contiene il valore relativo dipende solo dal numero di pixel rappresentati in ogni byte del buffer del video. Potreste esprimere la relazione nel modo seguente:

```
PixelBitOffset = PixelsPerByte - (x MOD PixelsPerByte) - 1
```

Tuttavia, risulta più pratico rappresentare l'offset di bit di un pixel come maschera di bit piuttosto che come numero ordinale di bit. Questo può essere realizzato facilmente con

una tabella di consultazione (ad esempio, un'istruzione *XLAT* dell'assemblatore) o con un'istruzione di scorrimento logico (questo è il motivo per cui i Listati da 26 a 29 ritornano l'offset di bit come numero di bit da spostare).

```
                TITLE   'Listato 28'
                NAME    PixelAddrHGC
                PAGE    55,132

;
; Nome:         PixelAddrHGC
;
; Funzione:     Determina indirizzo del buffer del pixel nella modalità
;               grafica 720x348 Hercules
;
; Chiamante:    AX = coordinata y (0-347)
;               BX = coordinata x (0-719)
;
; Ritorna:      AH = maschera di bit
;               BX = offset del byte nel buffer
;               CL = numero di bit da spostare verso sinistra
;               ES = segmento del buffer del video
;
BytesPerLine    EQU     90
OriginOffset    EQU     0                       ; offset del byte di (0,0)
VideoBufferSegEQU       0B000h

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC PixelÅddrHGC
PixelAddrHGC    PROC    near

                mov     cl,bl                   ; CL := byte meno significativo
                                                ;  di x

                shr     ax,1                    ; AX := y/2
                rcr     bx,1                    ; BX := 8000h*(y&1) + x/2
                shr     ax,1                    ; AX := y/4
                rcr     bx,1                    ; BX := 4000h*(y&3) + x/4
                shr     bx,1                    ; BX := 2000h*(y&3) + x/8
                mov     ah,BytesPerLine
                mul     ah                      ; AX := BytesPerLine*(y/4)
                add     bx,ax                   ; BX := 2000h*(y&3) + x/8 +
                                                ;  BytesPerLine*(y/4)
                add     bx,OriginOffset         ; BX := offset del byte nel
                                                ;  buffer del video

                mov     ax,VideoBufferSeg
                mov     es,ax                   ; ES:BX := indirizzo del byte
                                                ;  del pixel
                and     cl,7                    ; CL := x & 7
                xor     cl,7                    ; CL := numero di bit da
                                                ;  spostare a sinistra
                mov     ah,1                    ; AH := maschera di bit
                                                ;  non spostato
```

```
                ret

PixelAddrHGC    ENDP

_TEXT           ENDS

                END
```

Listato 28. *Calcolo dell' indirizzo di un pixel nella modalità grafica Hercules.*

```
                TITLE   'Listato 29'
                NAME    PixelAddr10
                PAGE    55,132

;
; Nome:         PixelAddr10
;
; Funzione:     Determina indirizzo di buffer del pixel nelle modalità EGA e
;               VGA originali:
;                       16 colori 320x200
;                       16 colori 640x200
;                       16 colori 640x350
;                       monocromatica (4 colori) 640x350
;                       2 colori 640x480
;                       16 colori 640x480
;
; Chiamante:    AX = coordinata y
;               BX = coordinata x
;
; Ritorna:      AH = maschera di bit
;               BX = offset del byte nel buffer
;               CL = numero di bit da spostare verso sinistra
;               ES = segmento del buffer del video
;
BytesPerLine    EQU     80                      ; byte in una linea orizzontale
OriginOffset    EQU     0                       ; offset del byte di (0,0)
VideoBufferSegEQU       0A000h

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC PixelAddr10
PixelAddr10     PROC    near

                mov     cl,bl                   ; CL := byte meno
                                                ;  significativo di x

                push    dx                      ; mantiene DX

                mov     dx,BytesPerLine         ; AX := y * BytesPerLine
                mul     dx

                pop     dx
                shr     bx,1
                shr     bx,1
                shr     bx,1                    ; BX := x/8
```

```
                add     bx,ax                   ; BX := y*BytesPerLine + x/8
                add     bx,OriginOffset         ; BX := offset del byte nel
                                                ;  buffer del video

                mov     ax,VideoBufferSeg
                mov     es,ax                   ; ES:BX := indirizzo del byte
                                                ;  del pixel

                and     cl,7                    ; CL := x & 7
                xor     cl,7                    ; CL := numero di bit da
                                                   spostare a sinistra
                mov     ah,1                    ; AH := maschera di bit non
                                                ;  spostato
                ret

PixelAddr10     ENDP

_TEXT           ENDS

                END
```

Listato 29. *Calcolo dell'indirizzo di un pixel nelle modalità grafiche CGA e VGA.*

Abbiamo ora un esempio ad alto livello della trasformazione di coordinate di un pixel per la modalità grafica a 4 colori 320 per 200 del CGA. Come mostra la Figura 41a, ogni byte del buffer del video contiene quattro pixel. Con quattro pixel per byte, 80 byte di dati rappresentano una riga di 320 pixel. L'origine sullo schermo, cioè l'offset di byte della parte visualizzata del buffer, è 0, dal momento che il buffer del video del CGA contiene solo uno schermo completo di pixel.

```
int PixelsPerByte = 4;
int BytesPerRow = 80;

int OriginOffset = 0;
static int Masks[] = { 0xC0, 0x30, 0x0C, 0x03 };

int x,y senza segno;
int ByteOffset,BitMask senza segno;
/* intercalazione buffer (0 o 0x2000) */
ByteOffset = (y & 1) < 3;

/*offset dell'inizio riga */
ByteOffset += BytesPerRow * (y/2);

/* offset del byte sullo schermo */
ByteOffset += (x / PixelsPerByte) % BytesPerRow;

/* offset del byte nel buffer del video */
ByteOffset += OriginOffset;

BitMask = Masks[x % PixelsPerByte];
```

La stessa routine in linguaggio Assembler risulta molto più efficiente, in quanto tutte le operazioni aritmetiche possono essere effettuate nei registri e nei mezzi registri (vedere Listato 26). Inoltre, se sapete che il numero dei byte per riga di pixel è una costante, potete aumentare ulteriormente le prestazioni eseguendo la moltiplicazione e la divisione come sequenze di spostamenti di bit.

Ad esempio, nel Listato 30 la coordinata *y* viene moltiplicata per 320 tramite una serie di operazioni logiche di spostamento invece che tramite una singola istruzione *MUL*. La routine che ne risulta opera più velocemente di circa il 40 percento sul modello 30 del PS/2 basato su 8086 e di circa il 10 percento sul modello 60 del PS/2 basato su 80286. Questa ottimizzazione complica in qualche modo il codice Assembler, ma la velocità raggiunta ripaga dello sforzo: le routine a basso livello come quelle dei Listati da 26 a 30 possono essere eseguite molte migliaia di volte in una applicazione orientata alla grafica.

```
                TITLE   'Listato 30'
                NAME    PixelAddr13
                PAGE    55,132

;
; Nome:         PixelAddr13
;
; Funzione:     Determina indirizzo del buffer del pixel nella modalità
;                a 256 colori 320x200
;
; Chiamante:    AX = coordinata y (0-199)
;               BX = coordinata x (0-319)
;
; Ritorna:      BX = offset del byte nel buffer
;               ES = segmento del buffer del video
;


OriginOffset    EQU     0                       ; offset del byte di (0,0)
VideoBufferSegEQU       0A000h

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC PixelAddr13
PixelAddr13     PROC    near
                xchg    ah,al                   ; AX := 256*y
                add     bx,ax                   ; BX := 256*y + x
                shr     ax,1
                shr     ax,1                    ; AX := 64*y
                add     bx,ax                   ; BX := 320*y + x

                add     bx,OriginOffset         ; BX := offset del byte nel
                                                ;  buffer del video

                mov     ax,VideoBufferSeg
                mov     es,ax                   ; ES:BX := indirizzo del byte
                                                ;  del pixel
```

```
                ret

PixelAddr13     ENDP

_TEXT           ENDS

                END
```

Listato 30. *Calcolo dell' indirizzo di un pixel nella modalità a 256 colori 320 per 200.*

Scalatura delle coordinate dei pixel

Una caratteristica della maggior parte delle modalità grafiche IBM è che la risoluzione orizzontale si differenzia dalla risoluzione verticale dei pixel. Ad esempio, nella modalità 640 per 200, un tipico monitor a colori da 200 linee visualizza circa 70 pixel per pollice orizzontale, ma solo circa 30 pixel per pollice verticale.

Questa discrepanza complica la mappatura dei pixel nel buffer di visualizzazione per la locazione su schermo, come mostra la Figura 47. Ad esempio, nella modalità 640 per 200, una linea tracciata tra il pixel a (0,0) nell'angolo superiore sinistro dello schermo e il pixel a (100,100) ha una deviazione matematica di 1, quindi vi aspettereste che venisse visualizzata ad un'angolazione di 45 gradi rispetto ai bordi sinistro e superiore dello schermo. La linea visualizzata (linea *a*, Figura 47) viene invece "compressa" orizzontalmente.

La visualizzazione di una linea ad un'angolazione di 45 gradi richiede la rappresentazione in scala delle coordinate di pixel per tener conto della discrepanza nella risoluzione verticale ed orizzontale. Nella modalità 640 per 200, il fattore di rappresentazione orizzontale in scala è circa 2,4 (risoluzione orizzontale : risoluzione verticale). Nell'esempio, si calcolerebbero le coordinate-*x* dei punti terminali come 0 (0x2,4) e come 240 (100x2,4). La linea calcolata in scala (linea *b*, Figura 47), con i punti terminali a (0,0) e (240,100), appare ad un'angolazione di 45 gradi sullo schermo.

Dovete calcolare in scala le coordinate (*x,y*) di tutti i pixel di tutte le figure geometriche in tutte le modalità grafiche, a meno che, naturalmente, il fattore di scala non sia 1. In caso contrario, i quadrati appariranno come rettangoli ed i cerchi appariranno come ellissi. Inoltre, dovete regolare il fattore di scala per le risoluzioni orizzontale e verticale per ogni modalità grafica. La Figura 48 è una tabella dei rapporti di scala orizzontale/verticale per le modalità grafiche dei sottosistemi di visualizzazione IBM con monitor tipici.

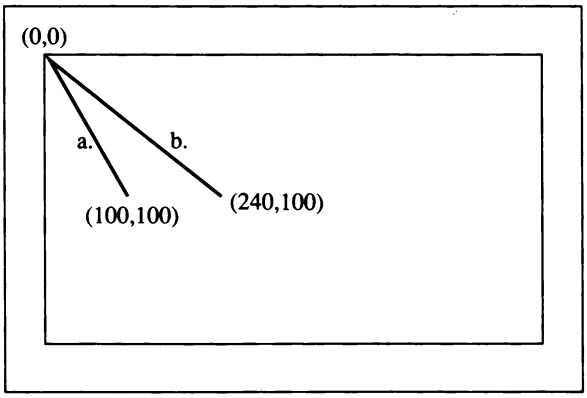


Figura 47. *Scalatura della coordinata dei pixel nella modalità grafica 640 per 200*

Numero modalità BIOS	Descrizione Modalità	Fattore di scala (orizzontale/verticale)
4,5	4 colori 320 per 200	1,20
6	2 colori 640 per 200	2,40
0DH	16 colori 320 per 200	1,20
0EH	16 colori 640 per 200	2,40
0FH	monocromatica 640 per 350	1,26 (monitor monocromat.)
10H	16 colori 640 per 350	1,37
11H	2 colori 640 per 480	1,00
12H	16 colori 640 per 480	1,00
13H	256 colori 320 per 200	2,40
	720 per 348 (Hercules)	1,43 (monitor monocromat.)

Figura 48. *Valori in scala dei pixel per le modalità grafiche dei PC e dei PS/2. Viene assunto un rapporto di aspetto di 1,33 (4:3) per i monitor a colori e 1,45 per i monitor monocromatici.*

Deformazione

Un problema di programmazione correlato è il rapporto dell'aspetto dello schermo e cioè il rapporto tra la larghezza dello schermo e la sua altezza. I monitor a colori utilizzati normalmente con i sottosistemi di visualizzazione IBM hanno rapporti di aspetto di circa 1,33 (4:3); per il normale monitor monocromatico, il rapporto di aspetto è di circa 1,45. Dal momento che lo schermo è rettangolare e non quadrato, la massima larghezza potenziale di un'immagine di schermo supera la propria altezza massima potenziale.

Questa limitazione deve essere sempre considerata quando si calcolano in scala le coordinate di pixel.

CONSIGLIO

Una funzione interessante dell'MCGA, della VGA e di altri sottosistemi di visualizzazione che offrono la risoluzione di 640 per 480 è che la risoluzione orizzontale e quella verticale sono identiche su un monitor con un rapporto di aspetto di 4:3. Potete pensare ai pixel in questa situazione come se fossero "quadrati". Con pixel "quadrati", la mappatura del buffer del video sullo schermo risulta più semplice in quanto il fattore di scala della coordinata di pixel è 1.

Attributi di visualizzazione dei pixel

In generale, i valori dei pixel determinano gli attributi di visualizzazione; in altre parole, i bit che rappresentano un pixel nel buffer del video determinano come il pixel apparirà sullo schermo. Il modo in cui i valori di pixel vengono decodificati nelle modalità grafiche è analogo al modo in cui vengono decodificati gli attributi alfanumerici. Nelle modalità grafiche, però, i valori di pixel possono variare da uno a otto bit, mentre gli attributi alfanumerici sono composti da quattro bit.

Adattatore colore/grafica

Nella modalità a 2 colori 640 per 200, ogni pixel viene rappresentato da un bit. Se il bit è impostato a 0, il pixel viene visualizzato in nero. Se il bit è impostato a 1, il pixel viene visualizzato con il colore specificato nei bit da 0 a 3 del registro di selezione colore del CGA (porta 3D9H). Questo è lo stesso registro che specifica il colore di sovrascansione nelle modalità alfanumeriche. Se modificate le modalità di visualizzazione programmando direttamente i registri del CRTC e i registri di controllo modalità del CGA, dovreste evitare colori di cornice o colori di pixel spuri programmando anche il registro di selezione colore.

Potete utilizzare la funzione 0BH di INT 10H per selezionare il colore di visualizzazione dei pixel non impostati a 0 nella modalità a 2 colori 640 per 200 (vedere Listato 31). Questa funzione del BIOS memorizza un valore di colore nel registro di selezione colore ed aggiorna la variabile CRT_PALETTE nell'area dei dati di visualizzazione a 0040:0066. Se aggirate il BIOS del video e programmate direttamente il registro di selezione colore, dovreste anche aggiornare CRT_PALETTE.

```
mov         ah,0Bh          ; AH := 0BH (numero funzione di INT 10H)
mov         bh,0            ; BH := numero di sottofunzione
mov         bl,ColorValue   ; BL := colore desiderato (0-0FH)
int         10h
```

Listato 31. *Colore di primo piano nella modalità grafica a 2 colori 640 per 200 del CGA.*

Nelle modalità a 4 colori 320 per 200, ogni pixel viene rappresentato da due bit, quindi i valori dei pixel possono variare da 0 a 3. I pixel con valore 0 vengono visualizzati con il valore di colore memorizzato nel registro di selezione colore alla porta 3D9H. Una particolarità del CGA è che il valore del registro di selezione colore determina sia il colore di sovrascansione (cornice) sia il colore per il valore di pixel 0. Ciò significa che non potete specificare un colore di cornice indipendentemente dal colore di sfondo con il CGA in questa modalità di visualizzazione.

I colori visualizzati per i pixel con valori diversi da zero vengono ricavati da una delle tre tavolozze hardware (vedere Figura 49). La tavolozza viene selezionata dai valori del bit 5 del registro di selezione colore (porta 3D9H) e del bit 2 del registro di controllo modalità alla porta 3D8H (Listato 32). Se il bit 2 del registro di controllo modalità è impostato a 1, la tavolozza comprende ciano, rosso e bianco. Se questo bit è impostato a 0, il bit 5 del registro di selezione colore seleziona o verde, rosso e giallo (se il bit 2 del registro di selezione colore è impostato a 0) o ciano, viola e bianco (se il bit 2 del registro di selezione colore è impostato a 1). In effetti, l'impostazione del bit 2 nel registro di selezione colore aggiunge il blu alla tavolozza; ciò significa che verde più blu produce il ciano, rosso più blu produce il viola e giallo più blu produce il bianco.

CONSIGLIO

L'impostazione a 1 del bit 2 del registro di controllo modalità del CGA disabilita la componente burst di colore del segnale di output su video composito dell'adattatore. Se utilizzate un monitor bianco e nero, vengono generate delle appropriate gradazioni di grigio per i quattro possibili valori di pixel quando il bit 2 viene impostato a 1.

Bit 2 del registro di controllo modalità = 0
Bit 5 del registro di selezione colore = 0

Valore del pixel	Colore visualizzato
1	Verde
2	Rosso
3	Giallo

(continua)

Bit 5 del registro di selezione colore = 1	
Valore del pixel	**Colore visualizzato**
1	Ciano
2	Viola
3	Bianco

Bit 2 del registro di controllo modalità = 1	
Valore del pixel	**Colore visualizzato**
1	Ciano
2	Rosso
3	Bianco

Figura 49. *Tavolozze disponibili nella modalità a 4 colori 320 per 200 del CGA.*

```
; ciano-rosso-verde

            mov     ax,40h
            mov     es,ax               ; ES := segmento dati BIOS del video
            mov     al,es:[65h]         ; AL := CRT_MODE_SET
            or      al,00000100b        ; bit 2 AL := 1
            mov     dx,3D8h             ; DX := porta di I/O controllo
                                        ; .modalità
            out     dx,al               ; aggiorna registro controllo modalità
            mov     es:[65h],al         ; aggiorna CRT_MODE_SET

; verde-rosso-giallo o ciano-viola-bianco

            mov     ax,40h
            mov     es,ax               ; ES := segmento dati BIOS del video
            mov     al,es:[65h]         ; AL := CRT_MODE_SET
            or      al,11111011b        ; bit 2 AL := 0
            mov     dx,3D8h             ; DX := porta di I/O controllo modalità
            out     dx,al               ; aggiorna registro controllo modalità
            mov     es:[65h],al         ; aggiorna CRT_MODE_SET

            mov     al,es:[66h]         ; AL := CRT_PALETTE
            and     al,11011111b        ; bit 5 AL := 0
            or      al,PaletteSelect ; 00000000b per verde-rosso-giallo
                                        ; 00100000b per ciano-viola-bianco
            inc     dx                  ; DX := porta di I/O di selezione
                                        ;  colore
            out     dx,al               ; aggiorna registro di selezione
                                        ;  colore
            mov     es:[66h],al         ; aggiorna CRT_PALETTE
```

Listato 32. *Tavolozze a quattro colori nella modalità a 4 colori 320 per 200 del CGA.*

Potete utilizzare le funzioni di INT 10H per scegliere una delle tre tavolozze a 4 colori. Il BIOS del video assegna due numeri di modalità video alla modalità grafica a 4 colori 320 per 200: nella modalità 4 del BIOS, il bit 2 del registro di controllo modalità è 0, e

in modalità 5, il bit 2 è impostato a 1. Di conseguenza, per selezionare la tavolozza ciano-rosso-bianco, usate la funzione 0 di INT 10H per impostare la modalità 5. Per selezionare le altre due tavolozze, usate la funzione 0 di INT 10H per impostare la modalità 4, quindi chiamate la funzione 0BH di INT 10H per scegliere o verde-rosso-giallo o ciano-viola-bianco, come mostrato nel Listato 33.

```
; ciano-rosso-bianco

            mov     ax,0005         ; AH := 0 (numero di funzione INT 10H)
                                    ; AL := 5 (modalità a 4 colori 320x200,
                                    ;  burst di colore disabilitato)
            int     10h

; verde-rosso-giallo o ciano-viola-bianco

            mov     ax,0004         ; AH := 0 (numero di funzione INT 10H)
                                    ;  AL := 4 (modalità a 4 colori 320x200,
                                    ;  burst di colore abilitato)
            int     10h

            mov     ah,0Bh          ; AH := numero di funzione INT 10H
            mov     bh,1
            mov     bl,PaletteID    ; 0 per verde-rosso-giallo
                                    ; 1 per ciano-viola-bianco
            int     10h
```

Listato 33. *Tavolozze a quattro colori nella modalità a 4 colori 320 per 200 del CGA utilizzando il BIOS del video.*

Potete selezionare i colori ad alta intensità nella tavolozza a 4 colori 320 per 200 impostando a 1 il bit 4 del registro di selezione colore. Quando questo bit è impostato a 0, gli stessi quattro colori vengono visualizzati con intensità normale.

Scheda grafica Hercules

Tutto si semplifica con un'HGC per quanto concerne gli attributi grafici. Nella modalità grafica monocromatica 720 per 348 su HGC e su HGC+, ogni pixel viene rappresentato da un bit. Se il bit è impostato a 1, il pixel viene visualizzato. Se il bit è impostato a 0, il pixel non viene visualizzato.

Adattatore grafico evoluto

Sebbene l'EGA supporti diverse modalità grafiche con valori di pixel che variano da 1 a 4 bit, esso decodifica i valori di pixel in un modo molto semplice. Come nelle modalità alfanumeriche, il valore di ogni pixel viene mascherato dal valore del registro di abilitazione piano colore del controller di attributo; il valore a 4 bit che ne risulta seleziona uno dei 16 registri di tavolozza del controller di attributo. Di conseguenza, l'attributo visualizzato di un pixel viene ricavato dal registro di tavolozza che corrisponde al valore di pixel.

Quando utilizzate la funzione 0 di INT 10H per selezionare una modalità video EGA, la routine del BIOS carica una serie di default di valori di colore nei registri di tavolozza (vedere Figura 50). I valori effettivi dipendono dalla modalità video, ma ogni serie si collega ai registri di tavolozza in modo che il colore visualizzato per un dato valore di pixel sia identico a quello visualizzato da un CGA. L'uso di questa funzione migliora la trasportabilità dei programmi tra il CGA e l'EGA, dal momento che un programma che non modifica mai i registri di tavolozza può girare con la stessa serie di colori su entrambi gli adattatori.

CONSIGLIO

I valori di default dei registri di tavolozza del BIOS per le modalità a 16 colori 320 per 200 e 640 per 200 sono corretti per i monitor a 200 linee, ma risultano errati per alcuni monitor compatibili EGA. Il monitor a colori evoluto dell'IBM converte i valori di default a 4 bit nelle modalità grafiche a 200 linee (vedere Figura 50) in valori di colore a 6 bit che emulano i 16 colori del CGA. Sfortunatamente, non tutti i monitor compatibili EGA si comportano in questo modo. Perciò, se utilizzate la funzione 0 di INT 10H per richiamare queste modalità (modalità numero 0DH e numero 0EH), generalmente dovreste programmare i registri di tavolozza con un'appropriata serie di valori, come ad esempio la serie di default usata nella modalità a 16 colori 640 per 350.

Modalità di emulazione CGA

Nella modalità a 2 colori 640 per 200, quando il bit 3 del registro di controllo modalità del controller di attributo (10H) è impostato a 0, un valore di pixel di 0 indica il registro di tavolozza 0, ed un valore di pixel di 1 indica il registro di tavolozza 1. Quando il bit 3 del controllo di modalità è impostato a 1, vengono utilizzati i registri di tavolozza 8 e 9. Con un monitor compatibile CGA, questi quattro registri di tavolozza possono contenere uno qualsiasi dei 16 valori di colore visualizzabili. Con un monitor a 350 linee compatibile EGA, questi registri possono contenere qualsiasi serie di quattro valori di colore tra i 64 visualizzabili.

Modalità a 16 colori 350 linee		
Registro di tavolozza	**Valore di colore**	**Attributo**
00H	00H	Nero
01H	01H	Blu a media intensità
02H	02H	Verde a media intensità
03H	03H	Ciano a media intensità
04H	04H	Rosso a media intensità
05H	05H	Viola a media intensità
06H	14H	Marrone
07H	07H	Bianco a media intensità
08H	38H	Bianco a bassa int. (grigio)
09H	39H	Blu ad alta intensità
0AH	3AH	Verde ad alta intensità
0BH	3BH	Ciano ad alta intensità
0CH	3CH	Rosso ad alta intensità
0DH	3DH	Viola ad alta intensità
0EH	3EH	Giallo ad alta intensità
0FH	3FH	Bianco ad alta intensità

Modalità a 16 colori 200 linee		
Registro di tavolozza	**Valore di colore**	**Attributo**
00H	00H	Nero
01H	01H	Blu
02H	02H	Verde
03H	03H	Ciano
04H	04H	Rosso
05H	05H	Viola
06H	06H	Giallo (marrone)
07H	07H	Bianco
08H	10H	Nero (grigio)
09H	11H	Blu ad alta intensità
0AH	12H	Verde ad alta intensità
0BH	13H	Ciano ad alta intensità
0CH	14H	Rosso ad alta intensità
0DH	15H	Viola ad alta intensità
0EH	16H	Giallo ad alta intensità
0FH	17H	Bianco ad alta intensità

Modalità grafica monocromatica 640 per 350		
Registro di tavolozza	**Valore di colore**	**Attributo**
00H	00H	Non visualizzato
01H	08H	Intensità normale
04H	18H	Evidenziato
05H	18H	Evidenziato

(continua)

Modalità grafica monocromatica 640 per 350		
Registro di tavolozza	**Valore di colore**	**Attributo**
08H	00H	Non visualizzato
09H	08H	Normale
0CH	00H	Non visualizzato
0DH	18H	Evidenziato

Figura 50. *Valori di default dei registri di tavolozza dell'EGA e della VGA.*

Nella modalità a 4 colori 320 per 200, ognuno dei quattro possibili valori di pixel (da 0 a 3) indica un corrispondente registro di tavolozza. Quando il bit 3 del registro di controllo modalità del controller di attributo è impostato a 0, vengono utilizzati i registri di tavolozza da 0 a 3; quando il bit 3 è impostato a 1, vengono utilizzati i registri di tavolozza da 8 a 0BH. Con un monitor compatibile CGA, potete memorizzare in questi registri di tavolozza qualsiasi combinazione di otto valori di colore dei 16 visualizzabili. Con un monitor compatibile EGA, potete memorizzare in questi registri qualsiasi combinazione di otto dei 64 valori di colore visualizzabili

In entrambe le modalità di emulazione CGA, il BIOS del video inizializza i registri di tavolozza con i valori di colore di default che corrispondono ai colori delle tavolozze hardware del CGA. Nella modalità a 2 colori 640 per 200, i colori di default sono nero, bianco e bianco evidenziato. Nelle modalità a 4 colori 320 per 200, il BIOS supporta le tavolozze verde-rosso-giallo e ciano-viola-bianco con intensità normali e alte.

Modalità a 16 colori

Nelle modalità a 16 colori 320 per 200, 640 per 200 e 640 per 350, ogni valore di pixel a 4 bit indica uno dei 16 registri di tavolozza. Per un monitor compatibile CGA, i registri di tavolozza possono contenere i soliti 16 colori, ma con un monitor compatibile EGA, potete specificare in ogni registro di tavolozza uno qualsiasi dei 64 colori visualizzabili.

Grafica monocromatica

Esistono due bit per pixel nella modalità grafica monocromatica 640 per 350 dell'EGA, quindi i valori di pixel possono variare da 0 a 3. Tuttavia, questa modalità grafica utilizza solo i piani di bit con numeri pari, quindi il controller di attributo dell'EGA interpreta solo i bit con numeri pari del solito valore di pixel a 4 bit. Di conseguenza, i bit 0 e 1 di un valore di pixel monocromatico a 2 bit indicano i bit 0 e 2 del corrispondente numero di registro di tavolozza a 4 bit (i bit 1 e 3 del numero di registro di tavolozza sono sempre impostati a 0). Perciò, i quattro possibili valori di pixel (0, 1, 2 e 3) in effetti si riferiscono rispettivamente ai registri di tavolozza 0, 1, 4 e 5 (vedere Figura 51).

Valore di pixel	Registro di tavolozza corrispondente
0(00B)	0(0000B)
1(01B)	1(0001B)
2(10B)	4(0100B)
3(11B)	5(0101B)

Figura 51. *Valori di pixel e registri di tavolozza nella modalità grafica monocromatica 640 per 350.*

CONSIGLIO

Su EGA con solo 64 KB di RAM video, i piani di bit dispari rappresentano i pixel agli indirizzi di buffer dispari, e i piani di bit pari rappresentano i pixel agli indirizzi di buffer pari (vedere Figura 45). In questa situazione, i valori di pixel nella modalità monocromatica 640 per 350 e in quella grafica a 4 colori 640 per 350 hanno una dimensione di due bit, ma i bit 0 e 2 vengono utilizzati per i pixel agli indirizzi pari, mentre i bit 1 e 3 vengono utilizzati per i pixel agli indirizzi di byte dispari.

I pixel monocromatici possono non essere visualizzati (valore 0 del registro di tavolozza), possono essere visualizzati ad intensità normale (08H) o possono essere visualizzati ad alta intensità (18H). La funzione 00H di INT 10H carica i registri di tavolozza con una serie di default di valori monocromatici ogni volta che selezionate la modalità video 0FH (vedere Figura 51).

Lampeggiamento

Nelle modalità grafiche originali dell'EGA (oltre che della VGA), i pixel possono assumere un attributo di lampeggiamento. Come nelle modalità alfanumeriche, si seleziona il lampeggiamento impostando a 1 il bit di abilitazione lampeggiamento del registro di controllo modalità del controller di attributo (bit 3 del registro 10H alla porta 3C0H). Nelle modalità a 16 colori, ciò fa sì che l'adattatore interpreti il bit più significativo (bit 3) di ogni valore di pixel a 4 bit come attributo di lampeggiamento, nello stesso modo in cui viene utilizzato il bit più significativo del byte di attributo di un carattere nelle modalità alfanumeriche. Di conseguenza, quando viene impostato il bit di abilitazione lampeggiamento, i pixel con valori da 8 a 0FH lampeggeranno e i pixel con valori da 0 a 7 non lampeggeranno. Nella modalità grafica monocromatica, tutti i pixel lampeggeranno a prescindere dal loro valore.

In ogni caso, l'EGA produce il lampeggiamento dei pixel in modo diverso nelle modalità grafiche rispetto a come vengono fatti lampeggiare i caratteri nelle modalità alfanumeriche. Nelle modalità grafiche, i pixel vengono fatti lampeggiare selezionando in modo alternato due diversi registri di tavolozza per il valore di ogni pixel. I due registri vengono indicati attivando e disattivando il bit 3 del valore di pixel alla velocità del lampeggiamento (circa due volte al secondo). Di conseguenza, i pixel vengono fatti lampeggiare alternando i valori dei primi otto registri di tavolozza (i registri da 00H a 07H) con i valori dei secondi otto (da 08H a 0FH).

Ad esempio, un pixel con un valore di 0AH viene fatto lampeggiare modificando ripetutamente il valore del bit 3 ogni volta che il bit di abilitazione lampeggiamento viene impostato. Perciò, il colore del pixel si alterna tra quello indicato dal registro di tavolozza 0AH (1010B) e quello del registro di tavolozza 02H (0010B). Se utilizzate la serie di registri di tavolozza di default del BIOS, questo pixel lampeggerà tra il verde e il verde ad alta intensità.

Una particolarità dell'attributo di lampeggiamento dell'EGA nelle modalità grafiche a colori è rappresentata da ciò che accade ai pixel con valori da 0 a 7, in pratica dove il bit 3 del valore di pixel è impostato a 0. Questi pixel non lampeggiano, ma vengono visualizzati come se il bit 3 fosse impostato a 1. Ad esempio, se usate i valori di tavolozza di default del BIOS, i pixel visualizzati all'intensità inferiore (valori di pixel da 0 a 7) diventano pixel non lampeggianti visualizzati ad alta intensità se si utilizzano i registri di tavolozza da 08H a 0FH.

Di conseguenza, nell'uso dell'attributo di lampeggiamento nelle modalità grafiche, dovreste riprogrammare i registri di tavolozza ogni volta che modificate il bit di abilitazione lampeggiamento, per mantenere una serie di colori coerente. Ad esempio, i valori di registro di tavolozza mostrati nella Figura 52 potrebbero rivelarsi utili in questo contesto. Questa tavolozza è stata studiata per essere utilizzata come alternativa alla tavolozza di default del BIOS (vedere Figura 50) quando il lampeggiamento viene abilitato. Se questa tavolozza viene utilizzata con il bit di abilitazione lampeggiamento impostato a 1, tutti i pixel ad alta intensità (i pixel con valore da 08H a 0FH) lampeggeranno, mentre tutti i pixel ad intensità normale non lampeggeranno.

Colore di cornice

Come nelle modalità alfanumeriche, potete impostare il colore di sovrascansione (cornice) memorizzando un valore di colore nel registro di colore di sovrascansione del controller di attributo (registro 11H, porta 3C0H). Le tecniche per impostare il colore di cornice sono spiegate nel Capitolo 3.

Registro di tavolozza	Valore di colore	Attributo
00H	00H	Nero (sfondo)
01H	39H	(colori ad alta intensità)
02H	3AH	(colori ad alta intensità)
03H	3BH	(colori ad alta intensità)
04H	3CH	(colori ad alta intensità)
05H	3DH	(colori ad alta intensità)
06H	3EH	(colori ad alta intensità)
07H	3FH	(colori ad alta intensità)
08H	00H	Nero (sfondo)
09H	01H	(colori a media intensità)
0AH	02H	(colori a media intensità)

(continua)

Registro di tavolozza	Valore di colore	Attributo
0BH	03H	(colori a media intensità)
0CH	04H	(colori a media intensità)
0DH	05H	(colori a media intensità)
0EH	14H	(colori a media intensità)
0FH	07H	(colori a media intensità)

Figura 52. *Valori dei registri di tavolozza per il lampeggiamento nella modalità a 16 colori 640 per 350.*

Scheda InColor Hercules

Sulla scheda InColor, il valore del bit 4 del registro di eccezioni (17H) determina se i registri di tavolozza vengono utilizzati per decodificare i valori di pixel, esattamente come avviene nelle modalità alfanumeriche. Quando questo bit è impostato a 1, ogni valore di pixel a 4 bit specifica un registro di tavolozza e il valore di colore a 6 bit del registro di tavolozza determina il colore visualizzato del pixel.

L'impostazione a 0 del bit 4 del registro di eccezioni aggira i registri di tavolozza. Ogni valore di pixel a 4 bit viene esteso a 6 bit duplicando il bit più significativo, e il valore risultante determina il colore. Questa procedura, chiamata estensione del segno, in effetti fa sì che il bit più significativo di un valore di pixel agisca come bit di "intensità", analogamente al modo in cui vengono decodificati gli attributi alfanumerici.

MCGA

L'MCGA emula le modalità grafiche del CGA e ne aggiunge due proprie, una modalità a 2 colori 640 per 480 e una modalità a 256 colori 320 per 200. La modalità a 256 colori è l'unica modalità video dell'MCGA che sfrutti pienamente il convertitore digitale-analogico (DAC).

Modalità grafiche a 2 colori

Gli attributi di pixel nelle modalità a 2 colori 640 per 200 e 640 per 480 sono dirette tramite i registri DAC del video. I pixel con il valore di 0 sono sempre mappati tramite il registro 0 del colore del DAC del video. Anche i pixel diversi da zero selezionano un predeterminato registro di colore del DAC del video, ma ciò avviene in uno dei due modi seguenti, a seconda del valore del bit 2 del registro di controllo modalità a 3D8H. Se il bit 2 è impostato a 1, viene selezionato il registro 7 del colore del DAC del video. Se il bit 2 è impostato a 0, i bit da 0 a 3 del registro di selezione colore (porta 3D9H) indicano un registro del DAC del video.

Su MCGA, il colore di sfondo nelle modalità grafiche a 2 colori non è necessariamente il nero come su CGA. In questo caso, invece, sia il primo piano sia lo sfondo possono essere di uno qualsiasi dei 256 K colori o dei 64 valori di scala di grigi che

l'MCGA può visualizzare. Usate la funzione 10H di INT 10H per impostare gli appropriati registri di colore del DAC del video.

CONSIGLIO

Quando il BIOS del video imposta le modalità grafiche a 2 colori, imposta a 0 il bit 2 del registro di controllo modalità e a 1111B (0FH) i bit da 0 a 3 del registro di selezione colore. Dal momento che i primi 16 registri di colore del DAC contengono i 16 colori disponibili su CGA, questa configurazione emula la configurazione di colore di default su CGA in modalità a 2 colori 640 per 200: i pixel dello sfondo vengono visualizzati in nero (il valore del registro di colore (0) del DAC del video) e i pixel di primo piano appaiono in bianco evidenziato (il valore del registro del colore 0FH del DAC del video).

Modalità grafica a 4 colori

L'MCGA emula fedelmente questa modalità grafica del CGA. La differenza principale è costituita dal fatto che l'MCGA mappa i quattro colori disponibili tramite i registri di colore del DAC del video esattamente come avviene nelle modalità grafiche a 2 colori. Di conseguenza, tutti i quattro colori possono essere selezionati a partire da 256 K possibilità che il DAC del video offre.

L'MCGA combina i bit 4 e 5 del registro di selezione colore (porta 3D9H) con il valore a 2 bit di ogni pixel per creare un valore a 4 bit che indichi uno dei primi 16 registri di colore del DAC del video (vedere Figura 53). Il BIOS del video inizializza i registri di colore del DAC del video con le tavolozze compatibili CGA. I colori vengono scelti in modo che il bit 5 del registro di selezione colore scelga le tavolozze verde-rosso-giallo e ciano-viola-bianco, e il bit 4 commuti tra tavolozze ad intensità normale e ad alta intensità, come avviene su CGA. Naturalmente, potete stabilire tavolozze a 4 colori completamente arbitrarie caricando diversi valori di colore nei registri di colore del DAC del video.

Bit 4 3D9H (intensità)	Valore di pixel Bit 1	Bit 0	Bit 5 3D9H (tavolozza)	Numero registro colore DAC del video	Colore di default
x	0	0	x	00H	Nero
0	0	1	0	02H	Verde
0	1	0	0	04H	Rosso
0	1	1	0	06H	Marrone

(continua)

Bit 4 3D9H (intensità)	Valore di pixel Bit 1	Bit 0	Bit 5 3D9H (tavolozza)	Numero registro colore DAC del video	Colore di default
1	0	1	0	0AH	Verde ad alta inten.
1	1	0	0	0CH	Rosso ad alta inten.
1	1	1	0	0EH	Giallo ad alta int.
0	0	1	1	03H	Ciano
0	1	0	1	05H	Viola
0	1	1	1	07H	Bianco
1	0	1	1	0BH	Ciano ad alta inten.
1	1	0	1	0DH	Viola ad alta inten.
1	1	1	1	0FH	Bianco ad alta int.

x = non significativo

Figura 53. *Valori di pixel e tavolozze nella modalità a 4 colori 320 per 200 dell'MCGA.*

Modalità grafica a 256 colori

Nella modalità a 256 colori, il valore di ogni pixel indica uno dei 256 registri di colore del DAC del video. Per selezionare un registro di colore del DAC del video, il valore di un pixel viene combinato (usando un AND logico) con il valore del registro di maschera del DAC del video (3C6H). Il valore che ne risulta seleziona un registro di colore del DAC (vedere Figura 46). Da momento che potete memorizzare uno qualsiasi dei 256K valori di colore in ogni registro di colore del DAC, potete visualizzare un'ampia gamma di toni e di intensità e creare delle immagini video piuttosto realistiche.Normalmente, quando viene selezionata la modalità a 256 colori 320 per 200, il BIOS del video programma i registri del DAC del video con una gamma di valori di colore di default (vedere Figura 54). I registri da 0 a 0FH contengono la gamma di default dei colori compatibili CGA. I registri da 10H a 1FH contengono una scala di grigi di intensità che varia gradualmente. I 216 registri successivi (da 20H a F7H) contengono tre gruppi di 72 colori, con il primo gruppo (i registri da 20H a 67H) ad alta intensità, il secondo (i registri da 68H a AFH) ad intensità media ed il terzo (i registri da B0H a F7H) a bassa intensità. Ogni gruppo di 72 colori è costituito da tre gamme di colori di saturazione decrescente (il colore tende al bianco); ogni gamma varia gradualmente nelle tonalità dal blu al rosso al verde.

CONSIGLIO

Per disabilitare e abilitare la programmazione del BIOS del video dei registri di colore del DAC, utilizzate la funzione 12H di INT 10H (vedere Appendice A).

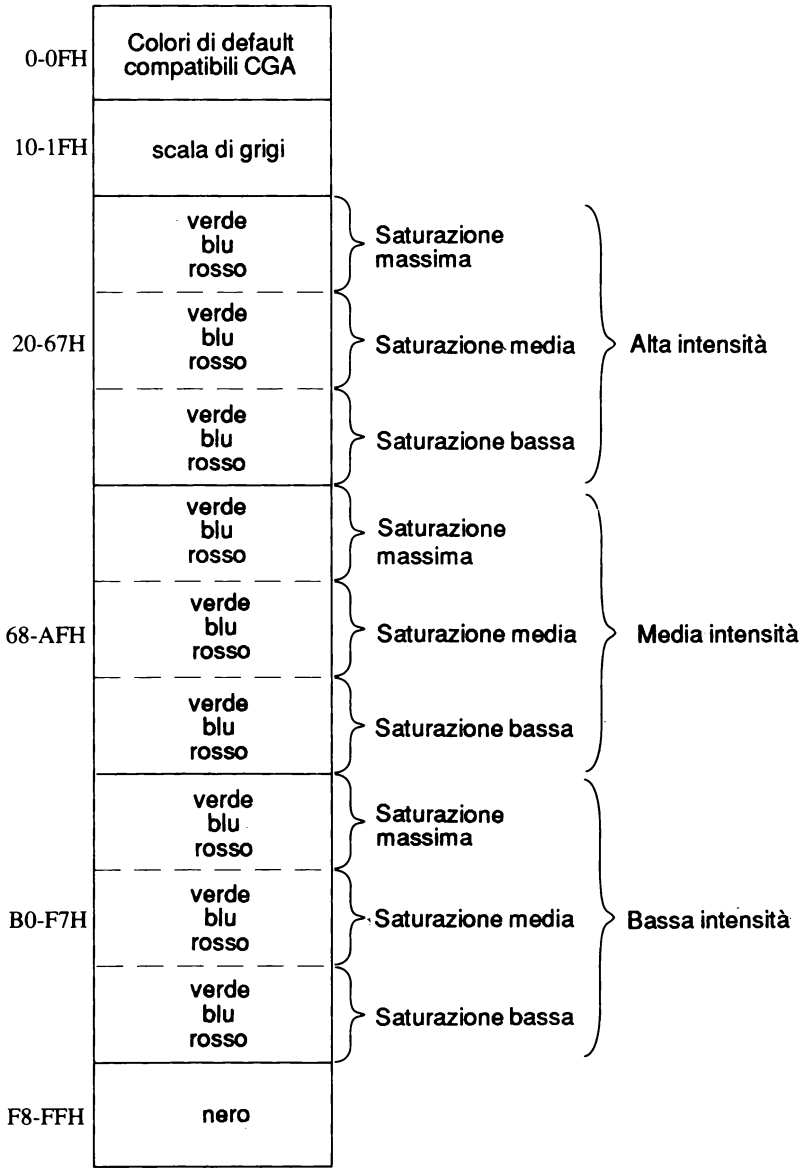


Figura 54. *Colori di default del DAC del video nella modalità a 256 colori 320 per 200 (MCGA e VGA).*

VGA

Come su EGA, i valori dei pixel della VGA vengono decodificati dal controller di attributo, utilizzando i registri di tavolozza, e quindi vengono trasferiti al DAC del video, seguendo la stessa logica applicata nelle modalità alfanumeriche (vedere Capitolo 3). Di conseguenza, il valore di un pixel seleziona il registro di tavolozza corrispondente; il valore del registro di tavolozza, insieme ai campi di bit del registro di selezione colore del controller di attributo, seleziona uno dei 256 registri di colore del DAC del video. Il DAC del video converte il valore RGB a 18 bit contenuto nei propri registri di colore nei corrispondenti segnali RGB analogici, che controllano il monitor.

L'unica eccezione a questo schema di decodifica di attributo si verifica nella modalità a 256 colori 720 per 320. In questa modalità, come su MCGA, ogni valore di pixel a 8 bit specifica direttamente uno dei 256 registri di colore del DAC del video, senza intermediazione del controller di attributo.

5
Programmazione dei pixel

Programmazione del piano di bit
EGA e VGA - Scheda InColor

Lettura del valore di un pixel
CGA - HGC e HGC+ - EGA
Scheda InColor - MCGA - VGA

Impostazione del valore di un pixel
CGA - HGC e HGC+ - EGA
Scheda InColor - MCGA - VGA

Riempimento del buffer del video
CGA - HGC e HGC+ - EGA e VGA
Scheda InColor - MCGA

Molte tecniche di programmazione grafica sono basate su routine che manipolano pixel individuali nel buffer del video. Questo capitolo illustra i punti fondamentali della programmazione dei pixel: la lettura del valore di un pixel, l'impostazione del valore di un pixel nel buffer del video e l'inizializzazione di un'area del buffer del video con una maschera di pixel.

Programmazione del piano di bit

Esiste una differenza fondamentale tra la programmazione in modalità grafica che usa i sottosistemi di visualizzazione la cui RAM video è organizzata in piani di bit paralleli (EGA, VGA e scheda InColor) e la programmazione in modalità grafica che usa gli altri sottosistemi di visualizzazione IBM. Su CGA, MCGA o adattatore monocromatico Hercules, il vostro programma accede ai pixel leggendo e scrivendo direttamente i byte nella RAM del video. Per contro, nelle modalità grafiche originali su EGA, VGA o scheda InColor, il vostro programma non può accedere direttamente alla RAM del video. Una speciale logica hardware del sottosistema di visualizzazione agisce da intermediario nell'operazione di accesso ai piani di bit.

I piani di bit della modalità grafica su EGA, VGA e schede InColor vengono indirizzati in parallelo; ciò significa che quando eseguite un'operazione di lettura o di scrittura di CPU ad un particolare indirizzo del buffer del video, l'indirizzo non si riferisce ad un byte, ma a quattro byte, uno per ogni piano di bit.

Quando eseguite un'istruzione 80x86 che tenta di leggere i dati da un indirizzo del buffer del video, quattro byte di dati vengono effettivamente spostati al di fuori del buffer. I dati però non vanno direttamente alla CPU. Al contrario, vengono copiati in una serie di latch a 8 bit. Ogni latch viene assegnato ad uno dei quattro piani di bit. L'esecuzione di un'operazione di lettura di CPU di 8 bit da un indirizzo nel buffer del video produce quindi il trasferimento di quattro byte (32 bit) di dati dal buffer del video ai latch (vedere Figura 55a). Le istruzioni come *MOV reg,mem,LODS,* e *CMP reg,mem* richiedono un'operazione di lettura della CPU, quindi provocano l'aggiornamento dei latch.

Analogamente, le istruzioni come *MOV mem,reg,STOS,* e *XOR mem,reg* provocano un'operazione di scrittura di CPU; in questo caso, tutti i quattro piani di bit possono essere aggiornati in parallelo utilizzando una combinazione dei dati dei latch, il byte dei dati che la CPU scrive, e un valore di pixel predefinito memorizzato in un registro di controllo grafica (vedere Figura 55b).

Alcune istruzioni di CPU richiedono sia un'operazione di lettura di CPU sia un'operazione di scrittura di CPU (la CPU legge un valore dalla memoria, effettua un'operazione su di essa e quindi riscrive il risultato in memoria). *MOVS* rappresenta un esempio ovvio, ma anche *OR mem,reg, AND mem,reg,* e *XOR mem,reg* generano un'operazione di lettura e di scrittura di CPU.

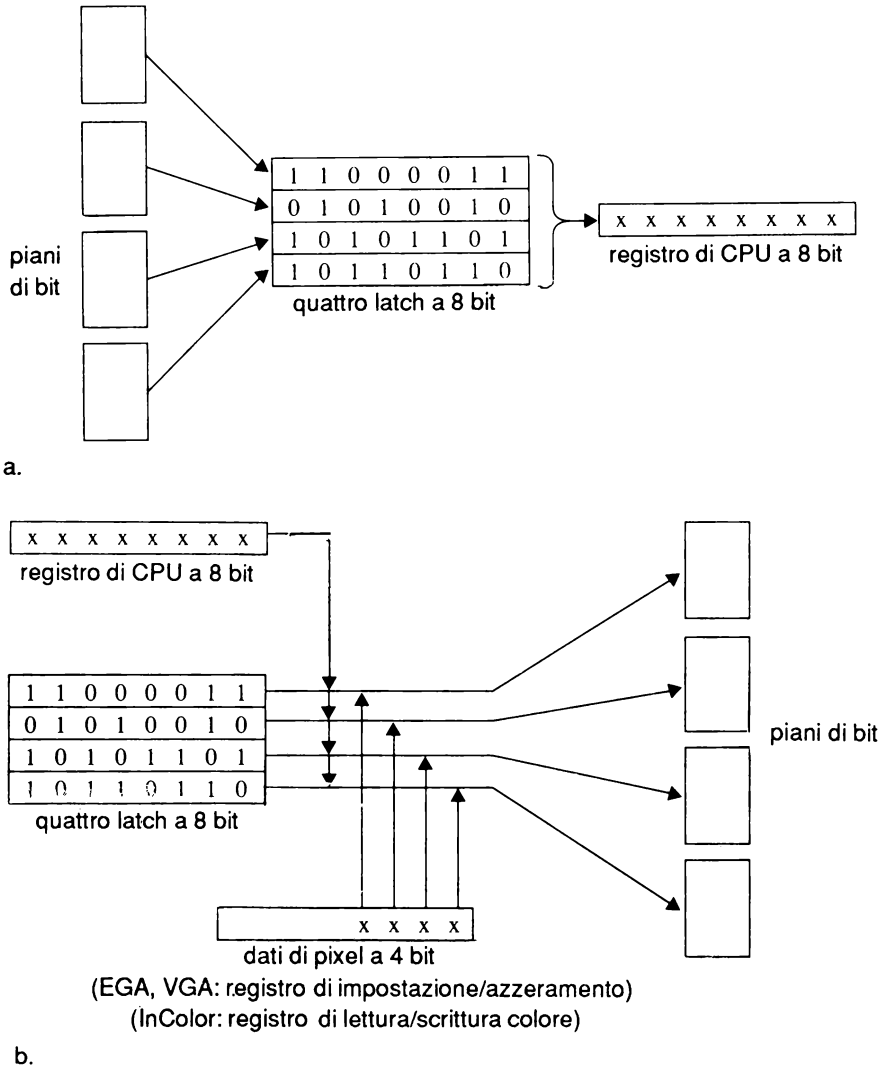


Figura 55. *Flusso dei dati di modalità grafica su EGA, VGA e scheda InColor durante operazioni di lettura (a.) e scrittura (b.) di CPU*

Quando un'istruzione di questo tipo fa riferimento ad un indirizzo della RAM del video, i latch vengono aggiornati durante l'operazione di lettura della CPU, quindi i piani di bit vengono aggiornati durante l'operazione di scrittura della CPU

L'uso dei latch per elaborare in parallelo i dati dei piani di bit permette di scrivere programmi illusoriamente semplici. Ad esempio, considerate il seguente frammento, che copia il secondo byte dei pixel nel buffer del video sul primo byte.

```
mov     ax,VideoBufferSegment
mov     ds,ax
mov     es,ax
mov     si,1                    ; DS:SI -> secondo byte
mov     di,0                    ; ES:DI -> primo byte
movsb
```

Questo programma sembra semplice. L'istruzione *MOVSB* apparentemente copia un byte dalla locazione di memoria a DS:SI sulla locazione a ES:DI, ma questo non è esattamente quello che avviene nelle modalità grafiche che utilizzano i piani di bit nel buffer del video dell'EGA, della VGA e della scheda InColor.

Ciò che accade effettivamente è quanto segue: l'istruzione *MOVSB* provoca un'operazione di lettura di CPU, seguita da un'operazione di scrittura di CPU. Dal momento che l'operazione di lettura di CPU fa riferimento ad un indirizzo del buffer del video, un byte di ogni piano di bit a quell'indirizzo viene caricato nei latch. Successivamente, dal momento che l'operazione di scrittura di CPU fa riferimento ad un indirizzzo del buffer del video, il contenuto dei latch viene copiato nei piani di bit all'indirizzo specificato. Di conseguenza, l'istruzione *MOVSB* provoca in effetti lo spostamento di quattro byte di dati invece che di uno.

CONSIGLIO

In questo esempio c'è di più di quello che si riesce a vedere. Considerate cosa accadrebbe se sostituiste l'istruzione *MOVSB* con l'istruzione *MOVSW*. Senza piani di bit e latch, ciò provocherebbe la copiatura di due byte di dati invece di uno. Tuttavia, metà dei dati di pixel verrebbero persi su EGA, VGA o scheda InColor. Questo perché l'istruzione *MOVSW* esegue in sequenza due operazioni di lettura di CPU a 8 bit, seguite da due operazioni di scrittura di CPU a 8 bit, quindi la seconda operazione di lettura di CPU aggiorna i latch prima che i byte del latch della prima operazione di lettura di CPU possano essere scritti.
Per questo motivo, dovreste usare istruzioni 80x86 a 16 bit con cautela quando accedete al buffer del video su EGA, VGA e scheda InColor. Istruzioni come *OR mem,reg, AND mem,reg* e *XOR mem,reg* noon funzionano correttamente con dati a 16 bit

I latch aumentano notevolmente l'efficienza nello spostamento dei dati dal/al buffer del video, ma il vero salto di qualità lo si ottiene nel trasferimento dei dati dai latch alla

CPU. Dal momento che i latch contengono 32 bit di dati e che un registro di CPU contiene solo otto bit, deve avvenire un qualche tipo di compressione dati durante la lettura di CPU. Per contro, nel trasferimento dei dati dalla CPU ai piani di bit, potete combinare il byte dei dati di CPU a 8 bit con il contenuto di tutti i quattro latch in diversi modi. La chiave di accesso alla programmazione in modalità grafica su EGA, VGA e su scheda InColor risiede nello sfruttamento delle trasformazioni dei dati che riguardano la CPU ed i latch.

EGA e VGA

Su EGA e VGA, il controller grafico gestisce tutti i trasferimenti di dati tra CPU, latch e buffer del video. Il controller grafico dell'EGA è costituito da due chip LSI: quello della VGA è incorporato nel chip VGA. Il controller grafico ha nove registri indirizzabili alla porta 3CFH tramite un registro di indirizzo alla porta 3CEH. I valori memorizzati nei registri controllano il modo in cui il controller grafico elabora i dati durante le operazioni di lettura e scrittura della CPU.

In un certo senso, il controller grafico permette di manipolare i dati di pixel di latch "bidimensionalmente". Alcune delle operazioni eseguibili sui dati di latch sono orientate ai byte ed hanno effetto separatamente su ogni latch. Altre operazioni sono orientate ai pixel in quanto considerano i dati di latch come se fossero una serie di otto valori di pixel; queste operazioni influenzano in modo separato ogni valore di pixel.

Il controller grafico può effettuare tre diverse operazioni orientate ai byte sui dati di latch. Esso può copiare il contenuto dei latch dal/al buffer del video; questa azione viene intrapresa implicitamente quando viene eseguita un'operazione di lettura o scrittura di CPU può ritornare il contenuto di uno dei latch ad un registro di CPU nel corso di un'operazione di lettura di CPU, e può anche combinare un byte di dati di un registro di CPU con i byte di uno o di tutti i latch durante una singola operazione di scrittura di CPU.

Il controller grafico elabora anche i dati di latch pixel per pixel. Durante un'operazione di lettura di CPU, il controller grafico può confrontare ogni valore di pixel di latch con un valore predefinito e ritornare il risultato del confronto alla CPU. Nel corso di operazioni di scrittura di CPU, può combinare un valore a 4 bit di CPU con uno o con tutti i valori di pixel dei latch; può utilizzare un valore a 8 bit di CPU come maschera che indichi quale degli otto pixel di latch vengono ricopiati nei piani di bit; inoltre, può combinare i valori di pixel dei latch con un valore predefinito a 4 bit.

Le operazioni orientate ai byte e le operazioni orientate ai pixel vengono entrambe programmate selezionando una modalità di scrittura e una modalità di lettura. Ogni modalità di scrittura imposta una sequenza predefinita di operazioni orientate ai byte e ai pixel che vengono effettuate quando viene eseguita un'operazione di scrittura di CPU. Analogamente, ogni modalità di lettura definisce una serie di azioni eseguite durante le operazioni di lettura di CPU. L'EGA ha tre modalità di scrittura e due modalità di lettura; la VGA oltre a queste cinque modalità ha un'ulteriore modalità di scrittura.

Fino a quando non vi diverranno familiari tutte le modalità di lettura e di scrittura del controller grafico, la loro ragione di essere potrebbe apparirvi oscura. In ogni caso, ogni modalità presenta vantaggi pratici in determinate situazioni di programmazione, come dimostrano gli esempi di questo e dei capitoli successivi.

Il registro di modalità del controller grafico (05H) contiene due campi di bit i cui valori specificano la modalità grafica di lettura e di scrittura. Ad esempio, per stabilire la modalità di lettura 1, imposterete a 1 il bit 3 del registro di modalità; per impostare la modalità di scrittura 2, memorizzerete il valore 2 (10B) nei bit 0 e 1 del registro di modalità (Listato 34).

```
mov     ax,0105h    ; AH := 1 (valore del reg 5)
                    ;  bit 3 := 0 (modalità di lettura 0)
                    ;  bit 0-1 := 1 (modalità di scrittura 1)
                    ; AL := numero di registro
mov     dx,3CEh     ; DX := porta del controller grafico
out     dx,ax
```

Listato 34. *Come impostare le modalità di lettura e scrittura del controller grafico. Questo esempio imposta la modalità di lettura 0 e la modalità di scrittura 1 nella modalità a 16 colori 640 per 350.*

CONSIGLIO

I valori di default del registro di modalità del controller grafico e degli altri registri del BIOS del video sono elencati nella Figura 56. E' consigliabile riportare i registri del controller grafico ai loro valori di default dopo averli modificati nel vostro programma.

Registro	Funzione	Valore
0	Impostazione/azzeramento	0
1	Abilitaz. impostazione/azzeramento	0
2	Confronto colore	0
3	Rotazione dati	0
4	Selezione mappa lettura	0
5	Modalità	Bit da 0 a 3 sempre a 0
6	Varie	(in relazione alla modalità video)
7	Colore ininfluente	0FH (modalità a 16 colori) 01H (modalità a 2 colori 640 per 480)
8	Maschera di bit	FFH

Figura 56. *Valori di default del BIOS ROM dei registri di controller grafico dell'EGA e della VGA.*

Modalità di lettura 0

Nella modalità grafica di lettura 0, il controller grafico ritorna il contenuto di uno dei quattro latch alla CPU ogni volta che un'operazione di lettura di CPU carica i latch (vedere Figura 57). Il valore del registro di selezione mappa di lettura (04H) indica quale latch deve essere letto. La modalità di lettura 0 permette quindi di leggere i byte di ogni piano di bit individuale; ciò risulta utile per il trasferimento dei dati tra i piani di bit e la RAM di sistema o un file su disco.

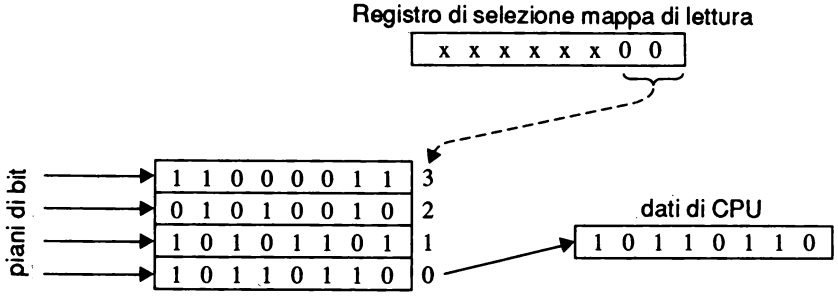


Figura 57. *Modalità grafica di lettura 0 dell'EGA e della VGA.*

Modalità di lettura 1

Nella modalità grafica di lettura 1, ognuno degli otto valori di pixel memorizzati nel latch nel corso di un'operazione di lettura di CPU viene confrontato con il valore del registro di confronto colore (02H). Il risultato del confronto viene ritornato alla CPU come singolo byte (vedere Figura 58). Quando un valore di pixel corrisponde al valore del registro di confronto colore, un bit del byte di dati della CPU viene impostato a 1; quando i valori sono diversi, il bit corrispondente del byte di dati viene impostato a 0.

Si noti come il valore del registro di colore ininfluente (07H) interagisce con il valore del pixel e con il valore di confronto colore. In effetti, l'impostazione a 0 di un bit del valore di colore ininfluente esclude un latch dal confronto. Ad esempio, un valore di confronto colore di 0111B provoca la partecipazione al confronto solo dei tre bit meno significativi di ogni valore di pixel. Un altro esempio: se memorizzate uno 0 nel registro di colore ininfluente, tutti i quattro bit del confronto diventano bit "ininfluenti" e quindi tutti i valori di pixel corrispondono al valore di confronto colore, e la CPU legge sempre il valore 11111111B in modalità di lettura 1.

Modalità di scrittura 0

La modalità grafica di scrittura 0 imposta una combinazione di operazioni orientate ai byte e ai pixel che vengono effettuate quando viene eseguita un'operazione di scrittura di CPU. Il byte di dati scritto dalla CPU può essere utilizzato per aggiornare uno o tutti i piani di bit; contemporaneamente, un valore di pixel predefinito può essere impiegato per aggiornare uno o tutti gli otto pixel coinvolti. Questo aggiornamento bidimensionale dei latch viene controllato in diversi modi utilizzando i valori dei registri di abilitazione im-

postazione/azzeramento, rotazione dati/selezione funzione e maschera di bit (vedere Figura 59).

Il registro di maschera di bit (08H) specifica come viene ricavato il nuovo valore di ognuno degli otto pixel del buffer del video. Quando un bit del registro di maschera di bit equivale a 0, il valore di pixel corrispondente viene copiato direttamente dai latch nel buffer del video. Per ogni bit 1 del valore di maschera di bit, il pixel corrispondente viene aggiornato con il valore di pixel del latch combinato con i dati di CPU o con il valore di pixel del registro impostazione/azzeramento. Di conseguenza, se un'operazione di scrittura di CPU segue immediatamente un'operazione di lettura allo stesso indirizzo, i soli pixel che vengono aggiornati sono quelli per i quali il bit corrispondente del registro di maschera di bit è impostato a 1.

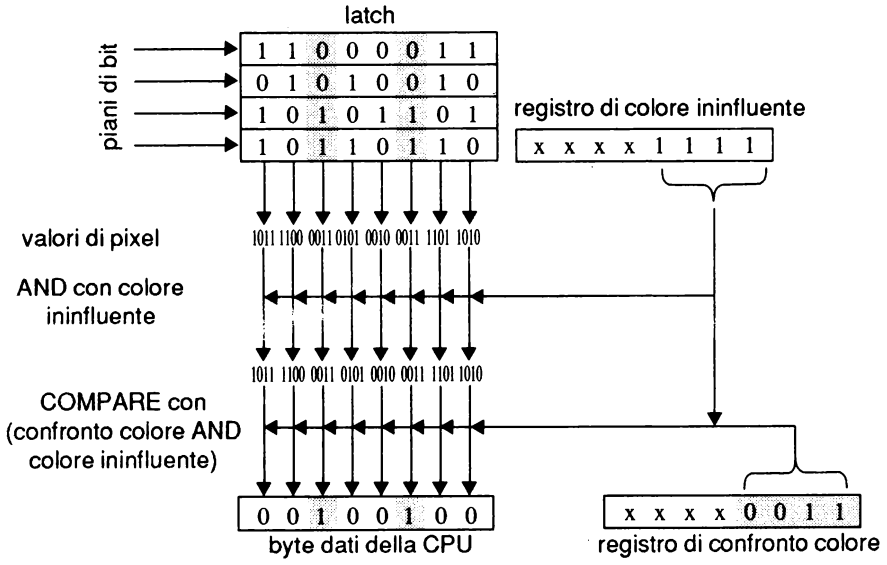


Figura 58. *Modalità grafica di lettura 1 dell'EGA e della VGA.*

Il registro di rotazione dati/selezione funzione (03H) contiene due campi di bit il cui contenuto influenza il modo in cui vengono aggiornati i pixel di latch. I bit 3 e 4 sono importanti in quanto il loro valore specifica quale operazione a logica binaria (AND, OR, XOR o sostituzione) viene utilizzata per aggiornare i pixel (vedere Figura 60). I bit da 0 a 2 specificano il numero di bit per il quale effettuare la rotazione verso destra del byte di dati della CPU prima di combinarlo con i dati del latch.

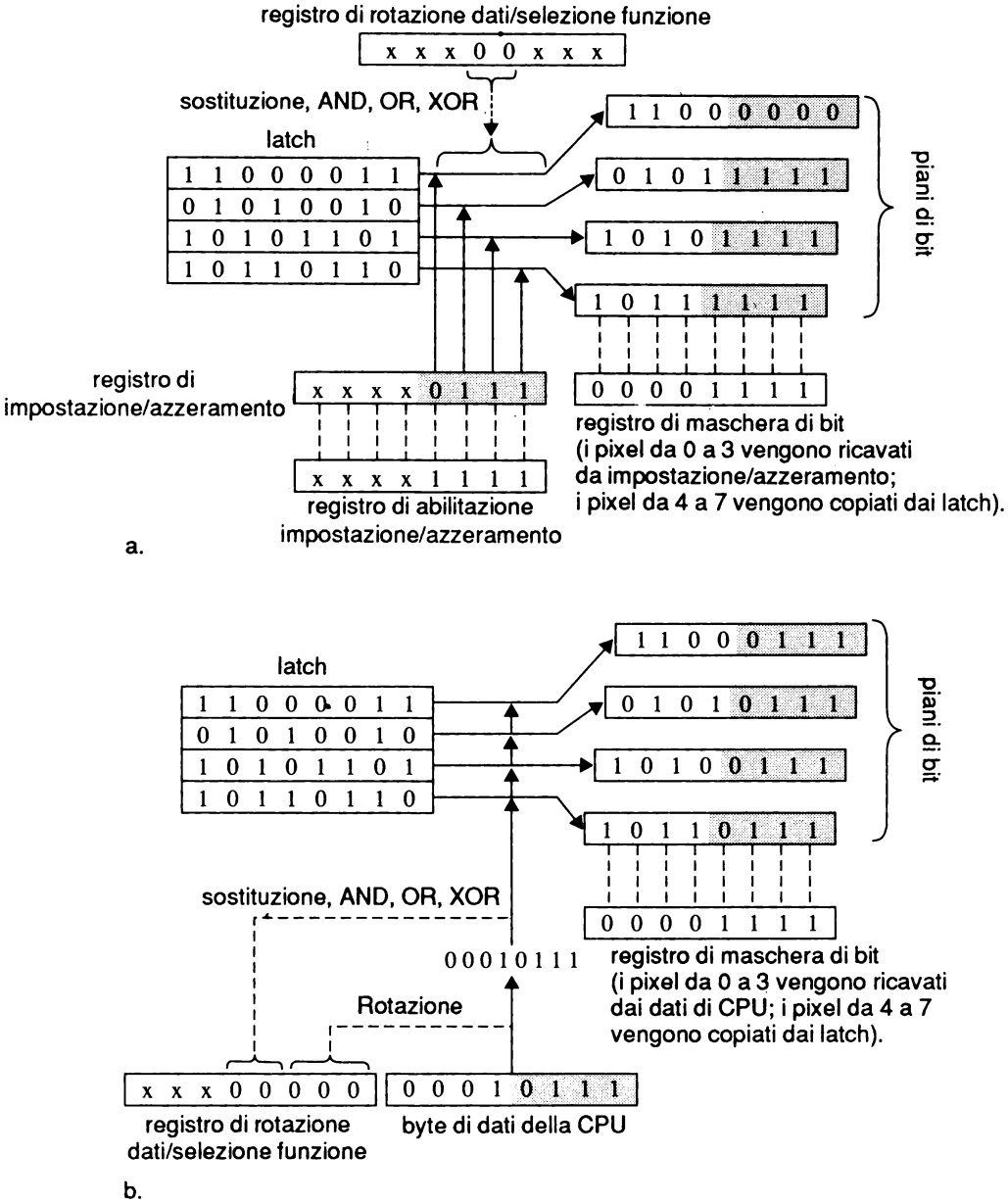


Figura 59. *Modalità grafica di scrittura 0 dell'EGA e della VGA: (a.) valore di abilitazione impostazione/azzeramento = 1111B, (b.) valore di abilitazione impostazione/azzeramento = 0000B.*

Valore di bit		Funzione
Bit 4	**Bit 3**	
0	0	Sostituzione
0	1	AND
1	0	OR
1	1	XOR

Figura 60. *Funzioni disponibili per l'aggiornamento dei pixel nelle modalità di scrittura 0, 2 e 3 dell'EGA e della VGA. I bit 3 . 4 del registro di rotazione dati/selezione funzione specificano quale funzione viene utilizzata*

CONSIGLIO

Questa capacità di rotazione dati non è particolarmente utile. In pratica, risulta generalmente più semplice permettere alla CPU di ruotare e di far scorrere i dati prima di scriverli nei piani di bit piuttosto di programmare il controller grafico in modo che effettui queste operazioni.

Il valore del registro di abilitazione impostazione/azzeramento (registro 01H) determina se i piani di bit vengono aggiornati byte per byte o pixel per pixel. Quando il valore di abilitazione impostazione/azzeramento è 0FH (1111B), ogni pixel viene aggiornato combinando il valore di pixel del latch con il valore del registro impostazione/azzeramento (registro 00H) utilizzando l'operazione logica specificata dal registro di rotazione dati/selezione funzione (fare riferimento alla Figura 59a). Quando il valore di abilitazione impostazione/azzeramento è 0, il byte di dati ruotato della CPU viene combinato con i byte di ogni latch, anche in questo caso utilizzando la funzione specificata dal registro di selezione funzione (vedere Figura 59b). In entrambi i casi vengono aggiornati solo i pixel mascherati dal registro di maschera di bit.

CONSIGLIO

Naturalmente, potete impostare il registro abilitazione impostazione/azzeramento su qualsiasi valore compreso tra 0 e 0FH. Ogni bit di ogni pixel viene quindi aggiornato combinandolo o con il bit corrispondente del registro impostazione/azzeramento o con il bit corrispondente del byte di dati della CPU a seconda del valore del bit corrispondente del registro di abilitazione impostazione/azzeramento. E' superfluo aggiungere che questo tipo di programmazione è ingannevole e viene usata raramente.

Modalità di scrittura 1

Nella modalità di scrittura 1, i latch vengono copiati direttamente sui piani di bit quando si verifica un'operazione di scrittura di CPU (vedere Figura 61). Né il valore del byte di dati della CPU né quelli dei registri rotazione dati/selezione funzione, maschera di bit, impostazione/azzeramento e abilitazione impostazione/azzeramento influenzano questo processo. Chiaramente, affinché un'operazione della modalità di scrittura 1 abbia

un senso, dovete per prima cosa eseguire un'operazione di lettura di CPU per inizializzare i latch.

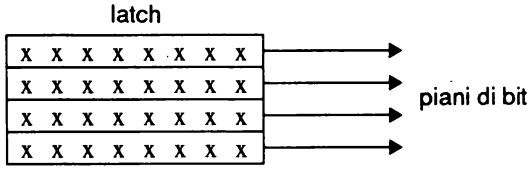


Figura 61. *Modalità grafica di scrittura 1 dell'EGA e della VGA.*

Modalità di scrittura 2

Nella modalità di scrittura 2, i bit meno significativi del byte scritto dalla CPU svolgono lo stesso ruolo del valore del registro di impostazione/azzeramento nella modalità di scrittura 0. Ciò significa che i piani di bit vengono aggiornati combinando i valori di pixel dei latch con i dati della CPU, utilizzando l'operazione logica specificata nel registro di rotazione dati/selezione funzione (vedere Figura 62). Come nella modalità di scrittura 0, il registro di maschera di bit specifica quali pixel vengono aggiornati utilizzando i valori di pixel combinati e quali pixel vengono aggiornati direttamente dai latch.

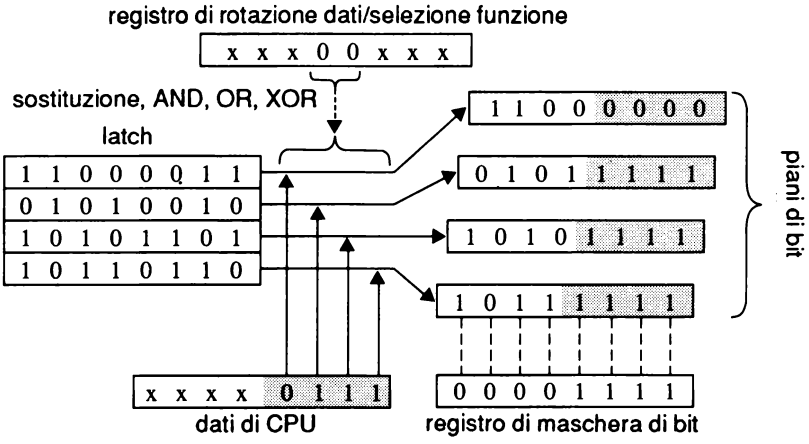


Figura 62. *Modalità grafica di scrittura 2 dell'EGA e della VGA.*

Modalità di scrittura 3

Nella modalità di scrittura 3 (supportata solo da VGA), i pixel vengono aggiornati combinando i valori di pixel dei latch con il valore del registro di impostazione/azzeramento. Anche in questo caso, il registro di rotazione dati/selezione funzione specifica l'operazione logica utilizzata per combinare i valori. Il byte di dati della CPU viene ruotato per il numero di bit indicato nel registro di rotazione dati/selezione funzione e combinato con il valore del registro di maschera di bit utilizzando un AND logico. La maschera di bit che ne risulta svolge quindi lo stesso ruolo del valore del registro di maschera di bit delle modalità di scrittura 0 e 2; ciò significa che determina quali pixel dei piani di bit vengono aggiornati combinando i valori di pixel dei latch con il valore di impostazione/azzeramento, e quali vengono invece aggiornati direttamente dai latch (vedere Figura 63).

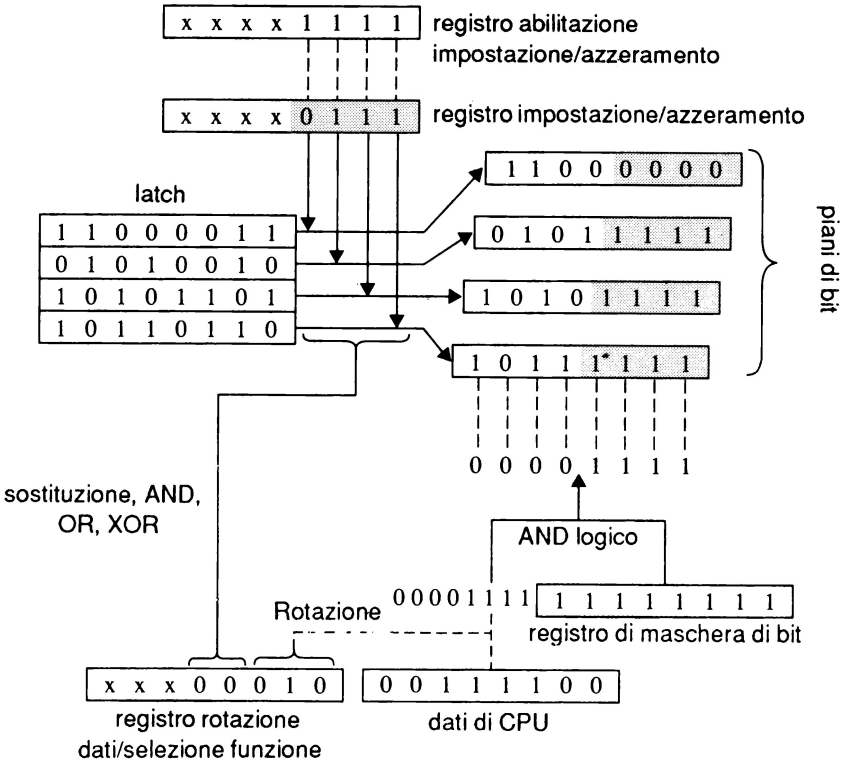


Figura 63. *Modalità grafica di scrittura 3 della VGA.*

Maschera di mappa del sequencer

Un livello supplementare di controllo è disponibile in tutte le modalità di scrittura del controller grafico dell'EGA e della VGA. Potete utilizzare il registro di maschera di mappa del sequencer (registro sequencer 02H) per abilitare o disabilitare in modo selettivo i trasferimenti di dati ai piani di bit. Nelle modalità grafiche a 16 colori, i bit da 0 a 3 di questo registro vengono normalmente impostati a 1 per permettere alle operazioni grafiche di scrittura di accedere a tutte le quattro mappe. Tuttavia, azzerando uno o più di questi bit, potete proteggere in scrittura le corrispondenti mappe di memoria.

Il registro di maschera di mappa del sequencer non viene utilizzato spesso, in quanto il controller grafico fornisce un controllo migliore per le operazioni orientate ai pixel. L'uso di questo registro si adatta maggiormente a tecniche come la stratificazione dei piani di bit (vedere Capitolo 12).

Scheda InColor

La scheda InColor ha due gate array, il codificatore e il decodificatore, che agiscono da intermediari negli accessi di CPU alla RAM del video. Il gate array del codificatore partecipa alle operazioni di scrittura su RAM video della CPU. Il gate array del decodificatore gestisce il trasferimento dei dati dalla RAM del video alla CPU, oltre che alla circuiteria di decodifica attributo della scheda.

La programmazione dell'interfaccia dell'hardware della modalità grafica della scheda InColor, compresi i chip del codificatore e del decodificatore, viene unificata tramite il registro di controllo della scheda impostato alle porte di I/O 3B4H e 3B5H (vedere Figura 64). Da un punto di vista software, non esiste distinzione tra il codificatore, il decodificatore e la circuiteria associata. I registri di controllo modalità grafica della scheda InColor sono analoghi ai registri di controllo dell'EGA e della VGA (vedere Figura 65).

Numero registro	Funzione del registro	Stato di lettura/scrittura
18H	Registro di maschera di piano	Solo scrittura
19H	Registro di controllo lettura/scrittura	Solo scrittura
1AH	Registro di colore lettura/scrittura	Solo scrittura
1BH	Registro di protezione latch	Solo scrittura

Figura 64. *Registri di controllo grafico sulla scheda InColor Hercules.*

InColor	EGA e VGA
Registro di maschera piano	Registro di maschera mappa del sequencer Registro di abilitazione piano colore del controller di attributo
Registro di controllo lettura/scrittura	Registro di modalità del controller grafico Registro di colore ininfluente del controller grafico
Registro di colore lettura/scrittura	Registro di impostazione/azzeramento del controller grafico
Registro di tavolozza	Registri di tavolozza del controller di attributo

Figura 65. *Registri di controllo funzionalmente simili su EGA, VGA e su scheda InColor.*

Come su EGA e VGA, gli accessi alla RAM del video nella modalità grafica vengono effettuati utilizzando una serie di quattro latch a 8 bit. Le operazioni di lettura e scrittura della CPU provocano il trasferimento in parallelo dei byte tra i latch e i corrispondenti piani di bit. Quando viene eseguita un'operazione di lettura di CPU, il decodificatore crea un latch di un byte per ogni piano di bit e ritorna un byte singolo di dati alla CPU. Quando viene eseguita un'operazione di scrittura di CPU, il codificatore combina i dati di latch con i valori di pixel memorizzati nel registro di colore lettura/scrittura e aggiorna i piani di bit con il risultato ottenuto.

Come l'EGA e la VGA, la scheda InColor può elaborare i dati della CPU ed i dati di latch in diversi modi. La scheda supporta quattro modalità grafiche di scrittura (vedere Figura 66), selezionate dai bit 4 e 5 del registro di controllo lettura/scrittura (19H). Esiste solo una modalità grafica di lettura, che è simile alla modalità di lettura 1 dell'EGA e della VGA.

Modalità di scrittura	Bit di dati di CPU = 0	Bit di dati di CPU = 1
0	Valore di sfondo	Valore di primo piano
1	Latch	Valore di primo piano
2	Valore di sfondo	Latch
3	NOT latch	Latch

Figura 66. *Sorgente di dati di pixel nelle modalità grafiche di scrittura di InColor.*

Modalità di scrittura da 0 a 3

In tutte le quattro modalità grafiche di scrittura, i dati di CPU funzionano come maschere a 8 bit. Il codificatore usa il valore di ogni bit della maschera per determinare come aggiornare il corrispondente valore di pixel nei latch. Ciò significa che la sorgente del

valore di pixel ad una particolare posizione di bit viene determinata dal valore del bit corrispondente nel byte di dati della CPU.

Ad esempio, nella modalità grafica di scrittura 1, quando un bit del byte dei dati di CPU è impostato a 1, il pixel corrispondente nel buffer del video viene sostituito dal valore di primo piano del registro di controllo lettura/scrittura; quando un bit del byte di dati della CPU è impostato a 0, il valore di pixel corrispondente viene copiato dai latch. Ad esempio, nella Figura 67, i pixel corrispondenti ai bit da 0 a 3 vengono sostituiti dal valore di primo piano del registro di controllo lettura/scrittura, mentre i restanti pixel vengono aggiornati dai valori di pixel dei latch.

Analogamente, nelle altre tre modalità grafiche di scrittura, il valore di ogni bit del byte di dati di CPU controlla come viene aggiornato il pixel corrispondente. Le modalità di scrittura si differenziano solo nel modo in cui vengono ricavati i valori di pixel (vedere Figura 66). Nella modalità di scrittura 0, il valore di primo piano o il valore di sfondo del registro di controllo lettura/scrittura sostituisce i pixel dei piani di bit. Nella modalità di scrittura 2, per ogni bit 0 del byte di dati della CPU, il valore di sfondo del registro di controllo lettura/scrittura viene utilizzato per aggiornare il pixel corrispondente nei piani di bit. Nella modalità di scrittura 3, ogni bit 0 del byte di dati della CPU provoca la sostituzione del pixel corrispondente nel buffer del video con il valore di pixel dei latch che risponde alla condizione logica NOT.

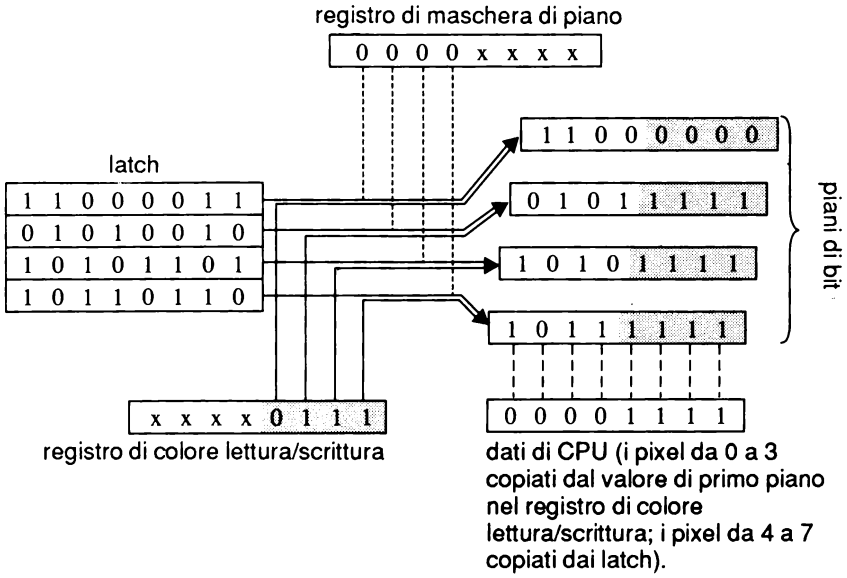


Figura 67. *Modalità grafica di scrittura 1 di InColor.*

Le operazioni di scrittura di CPU influenzano solo quei piani di bit specificati nel registro di maschera di piano (18H). La funzione di questo registro è quindi analoga a quel-

la del registro di maschera di mappa del sequencer dell'EGA. I bit da 4 a 7 di questo registro controllano quali dei quattro piani di bit sono modificabili; l'impostazione a 1 di questi bit impedisce l'aggiornamento dei piani di bit corrispondenti durante le operazioni di scrittura di CPU.

Modalità di lettura

La scheda InColor ha solo una modalità grafica di lettura (vedere Figura 68). Questa modalità è simile alla modalità di lettura 1 dell'EGA e della VGA. Quando viene eseguita un'operazione di lettura di CPU, i latch vengono caricati con i dati dei piani di bit. A differenza dell'EGA e della VGA, però, la scheda InColor permette di controllare quali valori individuali di pixel vengono trasferiti nei latch nel corso di un'operazione di lettura di CPU. Il valore di maschera di bit del registro di protezione latch (1BH) indica quali valori di pixel vengono trasferiti nei latch. Quando un bit del registro di protezione latch è impostato a 0, il valore di pixel corrispondente viene trasferito nel latch; quando un bit è impostato a 1, il valore di pixel corrispondente resta invariato.

Dopo che i valori di pixel specificati nei latch sono stati aggiornati dai piani di bit, il decodificatore confronta ogni valore di pixel dei latch con il valore di sfondo del registro di colore lettura/scrittura. Il risultato a 8 bit del confronto viene restituito alla CPU. Questa operazione è analoga alla modalità di lettura 1 dell'EGA e della VGA.

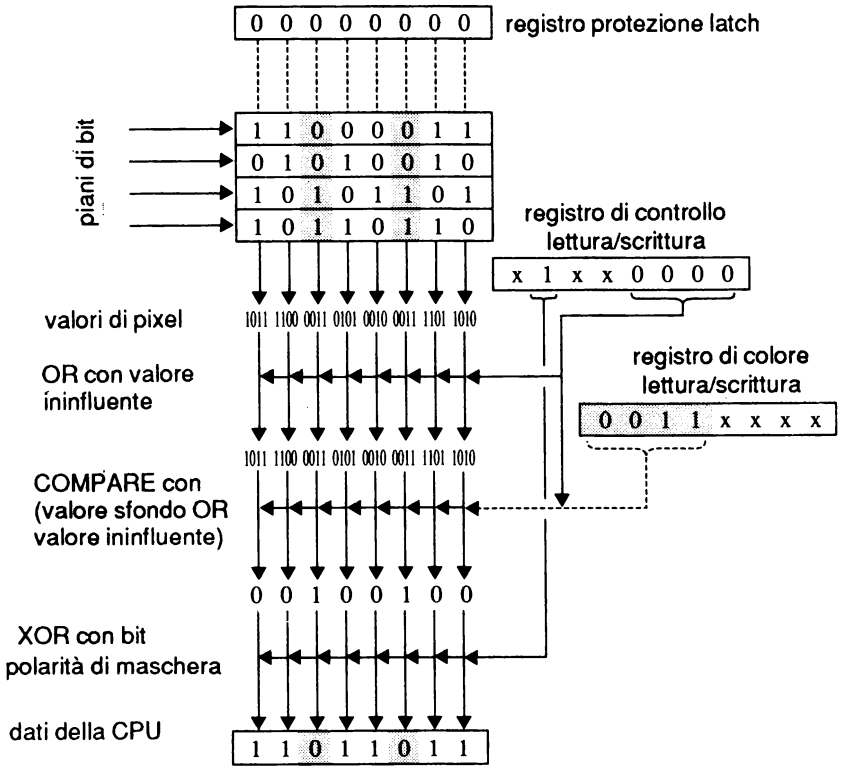


Figura 68. *Modalità di lettura della scheda InColor.*

Ibit da 0 a 3 del registro di controllo lettura/scrittura sono bit "ininfluenti" analoghi al valore di colore ininfluente dell'EGA e della VGA. L'impostazione a 1 di un bit "ininfluente" del controllo di lettura/scrittura ha l'effetto di escludere un latch dall'operazione di confronto con il valore di sfondo. Se impostate a 1 tutti i quattro bit "ininfluenti", tutti i valori di pixel corrisponderanno al valore di sfondo a prescindere dalla loro condizione

La polarità dei bit nel risultato restituita alla CPU dipende dal valore del bit di polarità di maschera (il bit 6 del registro di controllo lettura/scrittura). Quando questo bit è impostato a 0, i bit del risultato saranno 1, dove il valore di pixel dei latch corrisponde al valore di sfondo. L'impostazione a 1 del bit di polarità di maschera inverte il risultato; ciò significa che i bit sono impostati a 1 quando il valore di pixel dei latch non corrisponde al valore di sfondo.

Lettura del valore di un pixel

E' venuto il momento di passare a tecniche di programmazione specifiche per la manipolazione dei pixel sui vari sottosistemi di visualizzazione dei PC e dei PS/2. Una volta calcolati gli offset di byte e di bit di un particolare pixel all'interno del buffer del video, la determinazione del valore del pixel è solo una questione di isolare i bit che lo rappresentano nel buffer. Ciò vale sia per il CGA e l'HGC, che presentano una più semplice architettura di RAM video, sia per i più complicati sottosistemi di visualizzazione che impiegano i piani di bit.

CGA

Nella modalità a 2 colori 640 per 200, il valore di un pixel viene determinato semplicemente leggendo dal buffer del video il byte che contiene il pixel e verificando il valore del bit che rappresenta il pixel (vedere Listato 35).

```
                TITLE   'Listato 35'
                NAME    ReadPixel06
                PAGE    55,132

;
; Nome:         ReadPixel06
;
; Funzione:     Legge il valore di un pixel nella modalità a 2 colori 640x200
;
; Chiamante:    Microsoft C:
;
                        int ReadPixel06(x,y);
```

```
;                       int x,y;                /* coordinate di pixel */
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento di cornice
                                                ; di stack
ARGy            EQU     word ptr [bp+6]


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr06:near

                PUBLIC  _ReadPixel06
_ReadPixel06    PROC    near

                push    bp                      ; mantiene registri del chiamante
                mov     bp,sp

                mov     ax,ARGy                 ; AX := y
                mov     bx,ARGx                 ; BX := x
                call    PixelAddr06             ; AH := maschera di bit
                                                ; ES:BX -> buffer
                                                ; CL := # bit da spostare

                mov     al,es:[bx]              ; AL := byte contenente il pixel
                shr     al,cl                   ; sposta valore di pixel su bit
                                                ;  meno significativi
                and     al,ah                   ; AL := valore di pixel
                xor     ah,ah                   ; AX := valore di pixel

                mov     sp,bp                   ; ripristina registri
                                                ;  chiamante e ritorna
                pop     bp
                ret

_ReadPixel06    ENDP

_TEXT           ENDS

                END
```

Listato 35. *Determinazione di un valore di pixel nella modalità a 2 colori 640 per 200 del CGA.*

La tecnica per determinare il valore di un pixel nella modalità grafica a 4 colori 320 per 200, come mostrato nel Listato 36, è simile. Dopo aver isolato i bit che rappresentano il pixel, però, il vostro programma deve farli scorrere verso destra in modo che il valore ritornato rappresenti l'effettivo valore di pixel.

```
                TITLE   'Listato 36'
                NAME    ReadPixel04
                PAGE    55,132

;
; Nome:         ReadPixel04
;
; Funzione:     Legge il valore di un pixel nella modalità a 4 colori 320x200
;
; Chiamante:    Microsoft C:
;
;                       int ReadPixel04(x,y);
;
;                       int x,y;                /* coordinate di pixel */
;
ARGx            EQU     word ptr [bp+4]         ; indirizzamento cornice di stack
ARGy            EQU     word ptr [bp+6]


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr04:near

                PUBLIC  _ReadPixel04
_ReadPixel04    PROC    near

                push    bp                      ; mantiene registri chiamante
                mov     bp,sp

                mov     ax,ARGy                 ; AX := y
                mov     bx,ARGx                 ; BX := x
                call    PixelAddr04             ; AH := maschera di bit
                                                ; ES:BX -> buffer
                                                ; CL := # bit da spostare

                mov     al,es:[bx]              ; AL := byte contenente il pixel
                shr     al,cl                   ; sposta valore di pixel su
                                                ;  bit meno significativi
                and     al,ah                   ; AL := valore di pixel
                xor     ah,ah                   ; AX := valore di pixel

                mov     sp,bp                   ; ripristina registri
                                                ;  chiamante e ritorna
                pop     bp
                ret

_ReadPixel04    ENDP

_TEXT           ENDS

                END
```

Listato 36. *Determinazione di un valore di pixel nella modalità a 4 colori 320 per 200 del CGA.*

HGC e HGC+

L'unica differenza tra le routine di lettura pixel degli adattatori monocromatici Hercules e quelle usate nella modalità a 2 colori 640 per 200 del CGA è nel modo in cui viene calcolato l'indirizzo del pixel. Ad esempio, potete adattare la routine CGA mostrata nel Listato 35 all'HGC semplicemente sostituendo *PixelAddr06* con *PixelAddrHGC*.

EGA

Nelle modalità di emulazione CGA, le routine usate per il CGA funzionano senza dover essere modificate. Tuttavia, nelle modalità a 16 colori a 200 linee e nelle modalità a 350 linee, dovete programmare il controller grafico in modo che isoli i bit che rapresentano un pixel nei piani di bit del buffer del video, come avviene nella routine del Listato 37.

```
                TITLE   'Listato 37'
                NAME    ReadPixel10
                PAGE    55,132

;
; Nome:         ReadPixel10
;
; Funzione:     Legge il valore di un pixel nelle modalità grafiche EGA
;               originali
;
; Chiamante:    Microsoft C:
;
;                       int ReadPixel10(x,y);
;
;                       int x,y;                /* coordinate di pixel */
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento della
                                                ;  cornice di stack
ARGy            EQU     word ptr [bp+6]


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr10:near

                PUBLIC _ReadPixel10
_ReadPixel10    PROC    near

                push    bp                      ; mantiene i registri del chiamante
                mov     bp,sp
                push    si

                mov     ax,ARGy                 ; AX := y
```

```
                mov     bx,ARGx                 ; BX := x
                call    PixelAddr10             ; AH := maschera di bit
                                                ; ES:BX -> buffer
                                                ; CL := # bit da spostare

                mov     ch,ah
                shl     ch,cl                   ; CH := maschera di bit nella
                                                ;  posizione corretta

                mov     si,bx                   ; ES:SI - byte del buffer
                xor     bl,bl                   ; BL è usato per accumulare
                                                ;  il valore di pixel

                mov     dx,3CEh                 ; DX := porta del controller
                                                ;  grafico
                mov     ax,304h                 ; AH := numero iniziale di
                                                ;  piano di bit
                                                ; AL := numero reg. selezione
                                                ;  mappa lettura

L01:            out     dx,ax                   ; seleziona piano di bit
                mov     bh,es:[si]              ; BH := byte del piano di bit
                                                ;  corrente
                and     bh,ch                   ; mashera un bit
                neg     bh                      ; bit 7 di BH := 1 (se bit
                                                ;  mascherato = 1)
                                                ; bit 7 di BH := 0 (se bit
                                                ;  mascherato = 0)
                rol     bx,1                    ; bit 0 di BL := bit succ.
                                                ;  del valore di pixel
                dec     ah                      ; AH := numero piano di bit
                                                ;  successivo
                jge     L01

                mov     al,bl                   ; AL := valore di pixel
                xor     ah,ah                   ; AX := valore di pixel

                pop     si                      ; ripristina registri
                                                ;  chiamante e ritorna
                mov     sp,bp
                pop     bp
                ret

_ReadPixel10    ENDP

_TEXT           ENDS

                END
```

Listato 37. *Determinazione di un valore di pixel nelle modalità grafiche EGA originali.*

Questa routine utilizza la modalità di lettura 0 del controller grafico per leggere un singolo byte da ogni piano dell'EGA. Mano a mano che i byte vengono letti, i bit dei pixel desiderati vengono mascherati e concatenati per formare il valore del pixel.

CONSIGLIO

Nella modalità grafica monocromatica 640 per 350, vengono utilizzati solo i piani di bit 0 e 2 per rappresentare i valori di pixel. In queste modalità, solo i bit di questi due piani vengono concatenati per formare un valore di pixel (vedere Listato 38).
Come descritto nel Capitolo 4, le modalità grafiche 640 per 350 vengono mappate in modo diverso su un EGA dotato solo di 64 KB di RAM video rispetto a come vengono mappate su un EGA dotato di più memoria. Le mappe di memoria da 0 a 1 e da 2 a 3 vengono concatenate per formare due piani di bit. I pixel agli indirizzi di byte pari vengono rappresentati nelle mappe 0 e 2, mentre i pixel agli indirizzi di byte dispari vengono rappresentati nelle mappe 1 e 3.
Una routine per leggere i valori di pixel in queste modalità deve utilizzare l'indirizzo di byte del pixel per determinare quali mappe devono essere lette (vedere il Listato 39).

```
                TITLE   'Listato 38'
                NAME    ReadPixel0F
                PAGE    55,132

;
; Nome:         ReadPixel0F
;
; Funzione:     Legge il valore di un pixel nella modalità monocromatica
;               640x350
;
; Chiamante:    Microsoft C:
;
;                       int         ReadPixel0F(x,y);
;
;                       int x,y;                /* coordinate di pixel*/
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento di cornice
                                                ;  di stack
ARGy            EQU     word ptr [bp+6]


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr10:near

                PUBLIC  _ReadPixel0F
_ReadPixel0F    PROC    near

                push    bp                      ; mantiene registri chiamante
                mov     bp,sp
                push    si

                mov     ax,ARGy                 ; AX := y
```

```
                mov     bx,ARGx                 ; BX := x
                call    PixelAddr10             ; AH := maschera di bit
                                                ; ES:BX -> buffer
                                                ; CL := # bit da spostare

;concatena bit dei piani di bit 2 e 0

                mov     ch,ah
                shl     ch,cl                   ; CH := maschera di bit nella
                                                ;  posizione corretta
                mov     si,bx                   ; ES:SI - byte del buffer

                mov     dx,3CEh                 ; DX := porta del controller
                                                ;  grafico
                mov     ax,204h                 ; AH := numero iniziale del
                                                ;  piano di bit
                                                ; AL := numero reg. selezione
                                                ;  mappa lettura

                xor     bl,bl                   ; BL è usato per accumulare
                                                ;  il valore di pixel

L01:            out     dx,ax                   ; (come sopra)
                mov     bh,es:[si]
                and     bh,ch
                neg     bh

                rol     bx,1
                sub     ah,2
                jge     L01

                mov     al,bl
                xor     ah,ah

                pop     si
                mov     sp,bp
                pop     bp
                ret

_ReadPixel10F   ENDP

_TEXT           ENDS

                END
```

Listato 38. *Determinazione di un valore di pixel nella modalità grafica monocromatica dell'EGA.*

```
                TITLE   'Listato 39'
                NAME    ReadPixel10
                PAGE    55,132

; Nome:         ReadPixel10
;
; Funzione:     Legge il valore di un pixel nelle modalità 640x350 su EGA
                con 64K
;
```

```
; Chiamante:  Microsoft C:
;
;                   int        ReadPixel10(x,y);
;
;                   int x,y;             /* coordinate di pixel */
;

ARGx        EQU     word ptr [bp+4]      ; indirizzamento della cornice
                                         ;  di stack
ARGy        EQU     word ptr [bp+6]


_TEXT       SEGMENTbyte public 'CODE'
            ASSUME cs:_TEXT

            EXTRN   PixelAddr10:near

            PUBLIC  _ReadPixel10
_ReadPixel10 PROC   near

            push    bp                   ; mantiene registri chiamante

            mov     bp,sp
            push    si

            mov     ax,ARGy              ; AX := y
            mov     bx,ARGx              ; BX := x
            call    PixelAddr10          ; AH := maschera di bit
                                         ; ES:BX - buffer
                                         ; CL := # bit da spostare

; concatena bit dei piani di bit 2 e 0 (indirizzo byte pari)
;  o 3 e 1 (indirizzo byte dispari)

            mov     ch,ah
            shl     ch,cl                ; CH := maschera di bit nella

                                         ;  posizione corretta
            mov     si,bx                ; ES:SI - byte del buffer

            mov     ah,bl                ; AH := byte meno significativo
                                         ;  dell'indirizzo
            and     ax,100h              ; AH := byte meno significativo
                                         ;  dell'indirizzo
                                         ; AL := 0
            add     ax,204h              ; AH := numero iniziale piano
                                         ;  di bit (2 o 3)
                                         ; AL := numero reg. selezione
                                         ;  mappa lettura
            mov     dx,3CEh              ; DX := porta controller grafico
            xor     bl,bl                ; BL è usato per accumulare
                                         ;  il valore di pixel
L01:        out     dx,ax                ; (come sopra)
            mov     bh,es:[si]
            and     bh,ch
            neg     bh
            rol     bx,1
            sub     ah,2
```

```
                jge     L01

                mov     al,bl
                xor     ah,ah

                pop     si
                mov     sp,bp
                pop     bp
                ret

_ReadPixel10    ENDP

_TEXT           ENDS

                END
```

Listato 39. *Determinazione di un valore di pixel nelle modalità 640 per 350 su EGA con 64 KB.*

Scheda InColor

Come con EGA, la lettura del valore di un pixel su scheda InColor richiede la lettura separata di ogni piano di bit. Per fare ciò, dovete usare i bit "ininfluenti" del registro di controllo lettura/scrittura insieme al valore di sfondo del registro di colore lettura/scrittura per isolare il contenuto di ogni latch.

La routine del Listato 40 accumula il valore a 4 bit di un pixel concatenando un bit di ognuno dei quattro piani di bit della scheda InColor. La routine determina il contenuto di ogni piano di bit impostando il valore di sfondo nel registro di colore lettura/scrittura su 0FH (1111B) ed azzerando individualmente ogni bit "ininfluente" del registro di controllo lettura/scrittura. Quando viene eseguita ogni operazione di lettura di CPU (con l'istruzione AND CH,ES:[SI]), il valore restituito alla CPU è quindi il valore a 8 bit di uno dei quattro latch. Questo valore viene sottoposto ad un'operazione logica AND con la maschera di bit in CH, ed i bit isolati vengono accumulati in BL.

```
                TITLE   'Listato 40'
                NAME    ReadPixelInC
                PAGE    55,132

;
; Nome:         ReadPixelInC
;
; Funzione:     Legge il valore di un pixel nella modalità a 16 colori
;               720x348 InColor
;
; Chiamante:    Microsoft C:
;
;                       int         ReadPixelInC(x,y);
;
;                       int x,y;
```

```
ARGx            EQU     word ptr [bp+4]       ; indirizzamento cornice di stack
ARGy            EQU     word ptr [bp+6]

DefaultRWColorEQU       0Fh                   ; valore di default per registro
                                              ;  colore lettura/scrittura


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddrHGC:near

                PUBLIC _ReadPixelInC
_ReadPixelInC PROC      near

                push    bp                    ; mantiene registri chiamante

                mov     bp,sp
                push    si

                mov     ax,ARGy               ; AX := y
                mov     bx,ARGx               ; BX := x
                call    PixelAddrHGC          ; AH := maschera di bit
                                              ; ES:BX - buffer
                                              ; CL := # bit da spostare

; impostazione per esaminare separatamente ogni piano di bit

                mov     si,bx                 ; ES:SI - buffer

                shl     ah,cl
                mov     cl,ah                 ; CL := maschera di bit nella
                                              ;  posizione corretta

                mov     dx,3B4h               ; DX := porta controllo grafico

                mov     ax,0F01Ah             ; AH bit 4-7 := 1111b
                                              ;  (valore sfondo)

                                              ; AL := 1Ah (reg. colore
                                              ;  lettura/scrittura)
                out     dx,ax                 ; imposta valore sfondo

                mov     bx,800h               ; BH := 1000b (bit "ininfluenti"
                                              ;  iniziali)
                                              ; BL := 0 (valore iniziale per
                                              ;  risultato)
                dec     ax                    ; AL := 19h (numero registro
                                               controllo lettura/scrittura)
; loop attraverso i piani di bit aggiornando i bit "ininfluenti"

L01:            mov     ah,bh                 ; AH bit 0-3 := successivi bit
                                              ;  "ininfluenti"
                                              ; AH bit 6 := 0 (bit polarità
                                              ;  maschera)
                xor     ah,1111b              ; inverte bit "ininfluenti"
                out     dx,ax                 ; imposta registro controllo
                                              ;  lettura/scrittura
```

```
            mov     ch,cl                   ; CH := maschera di bit
            and     ch,es:[si]              ; piani bit di latch
                                            ; CH< > 0 se bit in latch è impostato

            neg     ch                      ; cf impostato se CH< >0
            rcl     bl,1                    ; accumula risultato in BL

            shr     bh,1                    ; BH := bit "ininfluenti" spostati
            jnz     L01                     ; loop fino a termine spostamento
                                            ;  di BH, quando BX = valore
                                            ;  del pixel
; ripristina stato default

            mov     ah,40h                  ; AH := valore default registro
                                            ;  controllo lettura/scrittura
            out     dx,ax

            inc     ax                      ; AL := 1Ah (numero registro
                                            ;  colore lettura/scrittura)
            mov     ah,DefaultRWColor
            out     dx,ax

            mov     ax,bx                   ; AX := valore del pixel

            pop     si                      ; ripristina registri chiamante
                                            ;  e ritorna
            mov     sp,bp
            pop     bp
            ret

_ReadPixelInC ENDP

_TEXT         ENDS

            END
```

Listato 40. *Determinazione di un valore di pixel nella modalità grafica di InColor.*

CONSIGLIO

Come avviene di solito nella programmazione dei piani di bit, la parte complicata di questo processo è l'impostazione dei valori del registro di controllo per produrre il risultato desiderato. Ad esempio, riportiamo di seguito cosa accade quando viene eseguita l'istruzione *AND CH,ES:[SI]*:

1. Un byte di ogni piano di bit viene copiato nei latch.

2. Ognuno degli otto pixel dei latch viene confrontato con il valore di sfondo (1111B), e gli otto bit che rispecchiano il risultato degli otto confronti vengono restituiti alla CPU. Dal momento che solo uno dei quattro bit "ininfluenti" del registro di controllo lettura/scrittura è impostato a 0, solo uno dei quattro bit di

ogni valore di pixel partecipal confronto. Se questo è il bit 1, il confronto risulta vero e il decodificatore ritorna un 1 nella posizione del bit corrispondente a questo valore di bit.

3. Gli otto bit restituiti alla CPU vengono sottoposti ad un'operazione logica AND con la maschera di bit in CH per fornire il risultato desiderato. Per una singola istruzione *AND* possiamo dire che vengono svolte molte azioni.

MCGA

Nelle modalità a 2 colori 640 per 200 e a 4 colori 320 per 200, le routine scritte per il CGA (mostrate nei Listati 35 e 36) funzionano anche su MCGA. Le altre due modalità grafiche dell'MCGA non danno ulteriori problemi (vedere i Listati 41 e 42), in quanto non utilizzano alcuna intercalazione di buffer come avviene invece nel caso delle modalità compatibili CGA, e in quanto non esistono piani di bit di cui preoccuparsi.

```
                TITLE   'Listato 41'
                NAME    ReadPixel11
                PAGE    55,132

;
; Nome:         ReadPixel11
;
; Funzione:     Legge il valore di un pixel nella modalità a 2 colori 640x480
;               (MCGA o VGA)
;
; Chiamante:    Microsoft C:
;
;                       int ReadPixel11(x,y);
;
;                       int x,y;                /* coordinate di pixel */
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento di cornice
ARGy            EQU     word ptr [bp+6]         ;  di stack


_TEXT           SEGMENT byte public 'CODE'
                ASSUME  cs:_TEXT

                EXTRN   PixelAddr10:near

                PUBLIC  _ReadPixel11
_ReadPixel11    PROC    near

                push    bp                      ; mantiene registri chiamante
                mov     bp,sp

                mov     ax,ARGy                 ; AX := y
```

```
                mov     bx,ARGx                 ; BX := x
                call    PixelAddr10             ; AH := maschera di bit
                                                ; ES:BX -> buffer
                                                ; CL := # bit da spostare

                mov     al,es:[bx]              ; AL := byte contenente il pixel
                shr     al,cl                   ; sposta valore di pixel su
                                                ;  bit meno significativi
                and     al,ah                   ; AL := valore di pixel
                xor     ah,ah                   ; AX := valore di pixel

                mov     sp,bp                   ; ripristina registri chiamante
                                                ;  e ritorna
                pop     bp
                ret

_ReadPixel11    ENDP

_TEXT           ENDS

                END
```

Listato 41. *Determinazione di un valore di pixel nella modalità a 2 colori 640 per 480 dell'MCGA e della VGA.*

```
                TITLE   'Listato 42'
                NAME    ReadPixel13
                PAGE    55,132

;
; Nome:         ReadPixel13
;
; Funzione:     Legge il valore di un pixel nella modalità a 256 colori
;               320x200 (MCGA e VGA)
;
; Chiamante:    Microsoft C:
;
;                       int ReadPixel13(x,y);
;
;                       int x,y;                /* coordinate di pixel */
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento di cornice
ARGy            EQU     word ptr [bp+6]         ;  di stack


_TEXT           SEGMENT byte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr13:near

                PUBLIC _ReadPixel13
_ReadPixel13    PROC    near

                push    bp                      ; mantiene registri chiamante
                mov     bp,sp
```

```
                mov     ax,ARGy             ; AX := y
                mov     bx,ARGx             ; BX := x
                call    PixelAddr13         ; ES:BX -> buffer

                mov     al,es:[bx]          ; AL := valore di pixel
                xor     ah,ah               ; AX := valore di pixel

                mov     sp,bp
                pop     bp
                ret

_ReadPixel13    ENDP

_TEXT           ENDS

                END
```

Listato 42. *Determinazione di un valore di pixel nella modalità a 256 colori 320 per 200 dell'MCGA e della VGA.*

VGA

Una volta scritte le routine di lettura pixel per CGA, EGA e MCGA, si sono viste le basi anche per quello che riguarda la VGA. L'unica modalità grafica VGA non disponibile sugli altri sottosistemi è la modalità a 16 colori 640 per 480. Tuttavia, la rappresentazione e l'indirizzamento dei pixel in questa modalità sono identici a quelli della modalità a 16 colori 640 per 350 dell'EGA, quindi potete utilizzare la routine del Listato 37 per entrambe le modalità.

Impostazione del valore di un pixel

In un certo qual modo, l'impostazione del valore di un pixel è il contrario della determinazione del suo valore. Una volta calcolati gli offset di bit e di byte di un particolare pixel, l'impostazione del suo valore è una semplice questione di introdurre i bit corretti nelle posizioni corrette del buffer del video.

Ciò che complica le routine di impostazione pixel è che potreste non desiderare sempre sostituire semplicemente il vecchio valore di un pixel con un nuovo valore. E' a volte auspicabile ricavare un nuovo valore di pixel eseguendo un'operazione logica binaria sul vecchio valore. Per questo motivo i controller grafici dell'EGA e della VGA supportano in modo diretto le operazioni logiche AND, OR e XOR sui valori di pixel, oltre che la sostituzione diretta dei vecchi valori con quelli nuovi.

CONSIGLIO

Dal momento che la gran parte delle operazioni di servizio di una routine di impostazione pixel è concentrata nel calcolo della locazione del pixel all'interno del buffer del video, potete mantenere il vostro programma limitato nelle dimensioni e modulare integrando diverse manipolazioni di valori di pixel in una singola routine piuttosto di scrivere routine separate per sostituire i pixel e per effettuare operazioni logiche binarie su di essi. Gli esempi di questo capitolo combinano queste diverse operazioni sui valori di pixel in routine unificate.
Dove l'operazione binaria richiede una subroutine diversa, l'indirizzo della subroutine viene memorizzato in una variabile (*SetPixelOp*). Questa tecnica risulta più flessibile della codifica di un salto all'operazione di pixel desiderata (sostituzione, AND, OR o XOR), in quanto potete cambiare l'indirizzo contenuto nella variabile con un'altra subroutine indipendente.
Gli esempi in questo capitolo non comprendono il codice per l'aggiornamento del valore di un pixel tramite esecuzione di un'operazione logica NOT. Potete utilizzare l'operazione XOR per ottenere lo stesso risultato di NOT senza diminuire le prestazioni e senza dover scrivere del codice aggiuntivo.

CGA

Per impostare un pixel nella modalità a 2 colori 640 per 200, mascherate il bit appropriato di un byte del buffer del video e quindi impostate il valore del bit. La routine del Listato 43 contiene quattro diversi metodi di impostazione del valore: sostituendo il vecchio valore di pixel con un nuovo valore e utilizzando le operazioni logiche OR, AND e XOR.

```
                TITLE   'Listato 43'
                NAME    SetPixel06
                PAGE    55,132

;
; Nome:         SetPixel06
;
; Funzione:     Imposta il valore di un pixel nella modalità a 2 colori 640x200
;
; Chiamante:    Microsoft C:
;
;                       azzera SetPixel(x,y,n);
;
;                       int x,y;                    /* coordinate di pixel */
;
;                       int n;                      /* valore di pixel */
;
```

```
ARGx            EQU     word ptr [bp+4]             ; indirizzamento di
                                                    ;  cornice di stack
ARGy            EQU     word ptr [bp+6]
ARGn            EQU     byte ptr [bp+8]


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr06:near

                PUBLIC  _SetPixel06
_SetPixel06     PROC    near

                push    bp                          ; mantiene registri
                                                    ;  del chiamante
                mov     bp,sp

                mov     ax,ARGy                     ; AX := y
                mov     bx,ARGx                     ; BX := x
                call    PixelAddr06                 ; AH := maschera del bit
                                                    ; ES:BX - buffer
                                                    ; CL := # bit da spostare
                                                    ;  a sinistra

                mov     al,ARGn                     ; AL := valore di pixel
                                                    ;  non spostato
                shl     ax,cl                       ; AH := maschera di bit
                                                    ;  nella posizione corretta
                                                    ; AL := valore di pixel
                                                    ;  nella osizione corretta

                jmp     word ptr SetPixelOp06       ; salta a routine di
                                                    ;  sostituzione,
                                                    ; AND, OR o XOR


                                                    ; routine per sostituire
                                                    ;  valore di pixel

ReplacePixel06:  not    ah                          ; AH := inverte maschera
                                                    ;  di bit
                and     es:[bx],ah                  ; azzera il valore di pixel
                or      es:[bx],al                  ; imposta il valore di pixel
                jmp     short L02



                                                    ; routine per effettuare
                                                    ;  AND del valore del pixel
ANDPixel06:     test    al,al
                jnz     L02                         ; non esegue nulla se
                                                    ;  il valore di pixel = 1

L01:            not     ah                          ; AH := inversione di
```

```
                                                ;  maschera di bit
                and     es:[bx],ah              ; imposta a 0 bit nel
                                                ;  buffer del video
                jmp     short L02
                                                ; routine per eseguire OR
                                                ;  del valore del pixel
ORPixel06:      test    al,al
                jz      L02                     ; non esegue nulla se
                                                ;  valore di pixel = 0

                or      es:[bx],al              ; imposta bit nel buffer
                                                ;  del video
                jmp     short L02


                                                ; routine per eseguire
                                                ;  XOR del valore del pixel
XORPixel06:     test    al,al
                jz      L02                     ; non esegue nulla se
                                                ;  valore di pixel = 0

                xor     es:[bx],al              ; esegue XOR del bit
                                                ;  nel buffer del video


L02:            mov     sp,bp                   ; ripristina reg.
                                                ;  chiamante e ritorna
                pop     bp
                ret

_SetPixel06     ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

SetPixelOp06    dw      ReplacePixel06          ; contiene indirizzo
                                                ;  dell'operazione del pixel

_DATA           ENDS

                END
```

Listato 43. *Impostazione di un valore di pixel nella modalità a 2 colori 640 per 200 del CGA.*

La routine per la modalità a 4 colori 320 per 200 è analoga. Questa routine, mostrata nel Listato 44, si differenzia dalla routine per la modalità a 2 colori 640 per 200 (vedere Listato 43) solo nella tecnica per il calcolo degli indirizzi dei pixel e nella rappresentazione dei pixel nei campi di bit che sono larghi due bit.

```
                TITLE   'Listato 44'
                NAME    SetPixel04
```

```
                PAGE    55,132

;
; Nome:         SetPixel04
;
; Funzione:     Imposta il valore di un pixel nella modalità a 4 colori 320x200
;
; Chiamante:    Microsoft C:
;
;                       azzera SetPixel(x,y,n);
;
;                       int x,y;                        /* coordinate di pixel */
;
;                       int n;                          /* valore di pixel */
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento di cornice
                                                ;  di stack
ARGy            EQU     word ptr [bp+6]
ARGn            EQU     byte ptr [bp+8]


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr04:near

                PUBLIC _SetPixel04
_SetPixel04     PROC    near

                push    bp                              ; mantiene registri del
                                                        ;  chiamante
                mov     bp,sp

                mov     ax,ARGy                         ; AX := y
                mov     bx,ARGx                         ; BX := x
                call    PixelAddr04                     ; AH := maschera di bit
                                                        ; ES:BX - buffer
                                                        ; CL := # bit da spostare
                                                        ;  a sinistra

                mov     al,ARGn
                shl     ax,cl                           ; AH := maschera di bit nella
                                                        ;  posizione corretta
                                                        ; AL := valore di pixel nella
                                                        ;  posizione corretta

                jmp     word ptr SetPixelOp04           ; salta a routine di
                                                        ;  sostituzione, AND, OR
                                                        ;  o XOR


                                                        ; routine per sostituire
                                                          valore di pixel

ReplacePixel04:not      ah                              ; AH := inverte maschera
```

```
                                                ;  di bit
                and     es:[bx],ah              ; azzera il valore di pixel
                or      es:[bx],al              ; imposta il valore di pixel
                jmp     short L02

                                                ; routine per effettuare
                                                ;  AND del valore del pixel

ANDPixel04:     not     ah                      ; AH := inversione di
                                                ;  maschera di bit
                or      al,ah                   ; AL := tutti i valori di
                                                ;  pixel eccetto gli 1
                and     es:[bx],al
                jmp     short L02


ORPixel04:      or      es:[bx],al              ; routine per eseguire OR
                                                ;  del valore del pixel
                jmp     short L02


XORPixel04:     xor     es:[bx],al              ; routine per eseguire
                                                ;  XOR del valore del pixel
L02:            mov     sp,bp                   ; ripristina reg. chiamante
                                                ;  e ritorna
                pop     bp
                ret

_SetPixel04     ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

SetPixelOp04    DW      ReplacePixel104         ;contiene indirizzo
                                                ; dell'operazione del Pixel


_DATA           ENDS

                END
```

Listato 44. *Impostazione di un valore di pixel nella modalità a 2 colori 320 per 200 del CGA.*

HGC e HGC+

Come potreste presupporre, una routine per la scrittura di un pixel nella modalità grafica monocromatica 720 per 348 dell'HGC può essere ricavata dalla routine equivalente della modalità a 2 colori 640 per 200 del CGA del Listato 43, sostituendo la routine di calcolo dell'indirizzo di pixel del CGA con quella dell'HGC (*PixelAddrHGC).*

EGA

Non è necessario preoccuparsi delle modalità di emulazione CGA (a 2 colori 640 per 200 e a 4 colori 320 per 200), in quanto le routine che funzionano su CGA funzionano ugualmente bene su EGA. Tuttavia, le cose si complicano nelle modalità grafiche originali dell'EGA. In queste modalità, esistono diversi metodi per programmare il controller grafico in modo che imposti il valore di un pixel individuale. Inoltre, la routine di impostazione pixel deve gestire correttamente le mappe di memoria video nelle modalità grafiche monocromatiche e nelle modalità a 4 colori 640 per 350 (su EGA con 64 KB).

Modalità di scrittura 0

Il metodo per l'impostazione del valore di un pixel nella modalità di scrittura 0 viene mostrato nel Listato 45. Per prima cosa, come al solito, si calcola l'offset del byte e la maschera di bit, che identificano la locazione del pixel all'interno del buffer del video. Successivamente si programma il controller grafico: si imposta la modalità di scrittura 0, si memorizza il valore di maschera di bit nel registro di maschera di bit, e si configurano i registri di impostazione/azzeramento e di abilitazione impostazione/azzeramento per il valore del pixel. Quindi si può eseguire un'operazione di lettura di CPU per effettuare i latch dei piani di bit, seguita da un'operazione di scrittura di CPU per copiare il contenuto dei latch ed il nuovo valore del pixel nei piani di bit.

```
                TITLE   'Listato 45'
                NAME    SetPixel10
                PAGE    55,132

;
; Nome:         SetPixel10
;
; Funzione:     Imposta il valore di un pixel nelle modalità grafiche EGA
;                originali.
;
;               *** Modalità di scrittura 0, impostazione/azzeramento ***
;
; Chiamante:    Microsoft C:
;
;                       azzera SetPixel(x,y,n);
;
;                       int x,y;                /* coordinate di pixel */
;
;                       int n;                  /* valore di pixel */
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento di cornice
                                                ;  di stack
ARGy            EQU     word ptr [bp+6]
ARGn            EQU     byte ptr [bp+8]

RMWbits         EQU     18h                     ; legge-modifica-scrive bit
```

```
_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN  PixelAddr10:near

                PUBLIC _SetPixel10
_SetPixel10     PROC   near

                push   bp                    ; mantiene registri chiamante
                mov    bp,sp

                mov    ax,ARGy               ; AX := y
                mov    bx,ARGx               ; BX := x
                call   PixelAddr10           ; AH := maschera di bit
                                             ; ES:BX - buffer
                                             ; CL := # bit da spostare verso
                                             ;  sinistra

; imposta registro maschera di bit del controller grafico

                shl    ah,cl                 ; AH := maschera di bit nella
                                             ;  posizione corretta
                mov    dx,3CEh               ; porta di registro di indirizzo GC
                mov    al,8                  ; AL := numero di registro
                                             ;  di maschera di bit
                out    dx,ax

; imposta registro modalità controller grafico

                mov    ax,0005h              ; AL := numero registro modalità
                                             ; AH := modalità di scrittura 0
                                             ; (bit 0,1)
                                             ;  modalità di lettura 0 (bit 3)
                out    dx,ax

; imposta registro rotazione dati/selezione funzione

                mov    ah,RMWbits            ; AH := legge-modifica-scrive bit
                mov    al,3                  ; AL := reg rotazione
                                             ;  dati/selezione funzione
                out    dx,ax

; imposta registri impostazione/azzeramento e abilitazione
; impostazione/azzeramento
                mov    ah,ARGn               ; AH := valore del pixel
                mov    al,0                  ; AL := numero reg
                                             ;  impostazione/azzeramento
                out    dx,ax

                mov    ax,0F01h              ; AH := valore per abilit.
                                             ;  impostaz./azzeram.(tutti i
                                             ;  piani di bit abilitati)
                                             ; AL := numero reg abilit.
                                             ;  impostaz./azzeram.
                out    dx,ax

; imposta il valore di pixel
```

```
                or      es:[bx],al              ; carica latch durante lettura
                                                ;  di CPU
                                                ;  aggiorna latch e piani di bit
                                                ;  duranteo perazione di
                                                ;  scrittura di CPU

; ripristina registri di default del controller grafico

                mov     ax,0FF08h               ; maschera di bit di default
                out     dx,ax

                mov     ax,0005                 ; registro di modalità di default
                out     dx,ax

                mov     ax,0003                 ; selezione funzione di default
                out     dx,ax

                mov     ax,0001                 ; abilitaz. impostazione/azzeram.
                                                ;  di default
                out     dx,ax

                mov     sp,bp                   ; ripristina registri chiamante
                                                ;  e ritorna
                pop     bp
                ret

_SetPixel10     ENDP

_TEXT           ENDS

                END
```

Listato 45. *Impostazione di un valore di pixel nelle modalità grafiche EGA originali utilizzando la modalità di scrittura 0.*

Si noti come il contenuto dei registri del controller grafico determini in che modo i piani di bit vengono aggiornati durante l'operazione di scrittura di CPU nell'istruzione *OR*. Il valore del registro di maschera di bit ha solo un bit non zero, quindi viene aggiornato solo un pixel. Questo pixel ricava il proprio valore dal registro impostazione/azzeramento (gli altri sette pixel vengono aggiornati dai latch; dal momento che l'operazione di lettura di CPU ha caricato i latch con gli stessi valori di pixel, l'operazione di scrittura di CPU non li modifica). Il valore di abilitazione impostazione/azzeramento è 1111B, quindi il byte di dati della CPU in AL non svolge alcun ruolo in questa operazione.

CONSIGLIO

Il BIOS dell'EGA dell'IBM usa la modalità di scrittura 0 per impostare i valori di pixel individuali nella funzione 0CH di INT 10H, ma la routine del BIOS non utilizza il registro di impostazione/azzeramento per specificare il valore del pixel. Al contrario, per prima cosa azzera il pixel utilizzando il registro di maschera di bit per isolarlo, quindi scrive un byte di dati di CPU

di 0. Successivamente il BIOS programma il registro di maschera di mappa del sequencer in modo che selezioni solo i piani di bit all'interno dei quali il valore di pixel desiderato contiene un bit non zero. La routine esegue quindi una seconda operazione di scrittura di CPU per impostare i bit non zero, come mostrato nel Listato 46.

Questa tecnica presenta due punti deboli: esistono metodi più semplici per ottenere lo stesso scopo, e la routine richiede del codice supplementare se desiderate sottoporre il valore di pixel del buffer del video alle operazioni logiche AND, OR o XOR. Per entrambe queste ragioni la funzione 0CH di INT 10H del BIOS del video è limitata sia per quanto riguarda la velocità sia per quanto riguarda la flessibilità.

```
                TITLE   'Listato 46'
                NAME    SetPixel10
                PAGE    55,132

;
; Nome:         SetPixel10
;
; Funzione:     Imposta il valore di un pixel nelle modalità grafiche EGA
;               originali.
;
;               *** Modalità di scrittura 0, maschera di mappa
;                   del sequencer ***
;
; Chiamante:    Microsoft C:
;
;                       azzera SetPixel(x,y,n);
;
;                       int x,y;                    /* coordinate del pixel */
;
;                       int n;                      /* valore del pixel */
;

ARGx            EQU     word ptr [bp+4]             ; indirizzamento di
                                                    ;  cornice di stack

ARGy            EQU     word ptr [bp+6]
ARGn            EQU     byte ptr [bp+8]


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr10:near

                PUBLIC _SetPixel10
_SetPixel10     PROC    near

                push    bp                          ; mantiene registri chiamante
```

```
            mov     bp,sp

            mov     ax,ARGy                     ; AX := y
            mov     bx,ARGx                     ; BX := x
            call    PixelAddr10                 ; AH := maschera di bit
                                                ; ES:BX - buffer
                                                ; CL := # bit da spostare
                                                ;  a sinistra

; imposta registro maschera di bit del controller grafico

            shl     ah,cl                       ; AH := maschera di bit
                                                ;  nella posizione corretta
            mov     dx,3CEh                     ; porta reg indir.
                                                ;  controller grafico
            mov     al,8                        ; AL := numero reg.
                                                ;  maschera di bit
            out     dx,ax

; azzera il valore di pixel

            mov     al,es:[bx]                  ; trasferisce in un latch
                                                ;  un byte di ogni piano
                                                ;  di bit
            mov     byte ptr es:[bx],0          ; azzera bit mascherati in
                                                ;  tutti i piani

; imposta registro maschera di mappa del sequencer

            mov     dl,0C4h                     ; DX := 3C4h (porta reg
                                                ;  indirizzos equencer)
            mov     ah,ARGn                     ; AH := valore reg. maschera
                                                ;  di mappa(selezione dei
                                                ;  bit non zero nel valore
                                                ;  di pixel abilita piani
                                                ;  di bit per sequencer)
            mov     al,2                        ; AL := numero reg.
                                                ;  maschera di mappa
            out     dx,ax

; imposta i bit non zero nel valore di pixel

            mov     byte ptr es:[bx],0FFh       ; imposta bit nei piani

                                                ;  di bit abilitati

; ripristina registri di default del sequencer

            mov     ah,0Fh                      ; AH := valore reg maschera
                                                ;  di mappa(tutti i piani
                                                ;  di bit abilitati)
            out     dx,ax

; ripristina registri di default del controller grafico

            mov     dl,0CEh                     ; DX := 3CEh (porta control.
                                                ;  grafico)
            mov     ax,0FF08h                   ; maschera di bit di default
```

```
                out     dx,ax

                mov     sp,bp                       ; ripristina reg.
                                                    ;  chiamante e ritorna
                pop     bp
                ret

_SetPixel10     ENDP

_TEXT           ENDS

                END
```

Listato 46. *Impostazione di un valore di pixel nelle modalità grafiche EGA originali utilizzando la maschera di mappa del sequencer.*

Modalità di scrittura 2

Un metodo in qualche modo più semplice per impostare il valore di un pixel è quello di utilizzare la modalità di scrittura 2. La routine del Listato 47 illustra questa tecnica. Come nella modalità di scrittura 0, il registro di maschera di bit determina il modo in cui ognuno degli otto pixel viene aggiornato. Nella modalità di scrittura 2, però, i nuovi valori di pixel vengono ricavati dalla combinazione del byte di dati della CPU con i valori di pixel dei latch; ciò evita di dover programmare i registri di impostazione/azzeramento e di abilitazione impostazione/azzeramento e porta quindi ad un codice più veloce e più breve.

```
                TITLE   'Listato 47'
                NAME    SetPixel10
                PAGE    55,132

;
; Nome:         SetPixel10
;
; Funzione:     Imposta il valore di un pixel nelle modalità grafiche EGA
;                originali.
;
;               *** Modalità di scrittura 2 ***
;
; Chiamante:    Microsoft C:
;
;                       azzera SetPixel(x,y,n);
;
;                       int x,y;        /* coordinate del pixel */
;
;                       int n;          /* valore del pixel */
;
ARGx            EQU     word ptr [bp+4]         ; indirizzamento della cornice
                                                ;  di stack
ARGy            EQU     word ptr [bp+6]
ARGn            EQU     byte ptr [bp+8]
```

```
RMWbits        EQU    18h            ; legge-modifica-scrive bit


_TEXT          SEGMENTbyte public 'CODE'
               ASSUME cs:_TEXT

               EXTRN  PixelAddr10:near

               PUBLIC _SetPixel10
_SetPixel10    PROC   near

               push   bp             ; mantiene cornice di stack
               mov    bp,sp

               mov    ax,ARGy        ; AX := y
               mov    bx,ARGx        ; BX := x
               call   PixelAddr10    ; AH := maschera di bit
                                     ; ES:BX - buffer
                                     ; CL := # bit da spostare verso sinistra

; imposta registro maschera di bit del controller grafico

               shl    ah,cl          ; AH := maschera di bit nella posizione
                                     ;  corretta
               mov    dx,3CEh        ; porta del registro di indirizzo GC
               mov    al,8           ; AL := numero del registro di
                                     ;  maschera di bit
               out    dx,ax

; imposta registro modalità controller grafico

               mov    ax,205h        ; AL :=  numero registro modalità
                                     ; AH :=  modalità di scrittura 2 (bit 0,1)
                                     ;        modalità di lettura 0 (bit 3)
               out    dx,ax

; imposta il registro rotazione dati/selezione funzione

               mov    ah,RMWbits     ; AH := legge-modifica-scrive bit
               mov    al,3           ; AL := reg rotazione dati/selezione
                                     ;  funzione
               out    dx,ax
; imposta valore del pixel

               mov    al,es:[bx]     ; trasferisce al latch un byte di ogni
                                     ;  piano  di bit
               mov    al,ARGn        ; AL := valore di pixel
               mov    es:[bx],al     ; aggiorna tutti i piani di bit

; ripristina registri di default del controller grafico

               mov    ax,0FF08h      ; maschera di bit di default
               out    dx,ax

               mov    ax,0005        ; registro di modalità di default
               out    dx,ax

               mov    ax,0003        ; selezione funzione di default
```

```
                out     dx,ax

                mov     sp,bp       ; ripristina cornice di stack e ritorna
                pop     bp
                ret

_SetPixel10     ENDP

_TEXT           ENDS

                END
```

Listato 47. *Impostazione di un valore di pixel nelle modalità grafiche EGA originali utilizzando la modalità di scrittura 2.*

Le routine dei Listati 45 e 47 sono state progettate per operare correttamente quando il registro di selezione funzione specifica la funzione AND, OR o XOR. Di conseguenza, non dovete scrivere del codice supplementare per effettuare queste manipolazioni di pixel alternative nelle modalità grafiche EGA originali.

Inoltre, se prestate attenzione nell'uso dei corretti valori di pixel, le routine dei Listati 45 e 47 possono essere utilizzate in qualsiasi modalità grafica EGA originale. Per assicurarsi che i bit appropriati delle mappe di memoria vengano aggiornati nella modalità monocromatica 640 per 350, usate solo i valori di pixel di 0, 1, 4 e 5. Su un EGA con 64 KB di RAM, usate i valori di pixel 0, 3, 0CH e 0FH.

Scheda InColor

La routine del Listato 48 aggiorna un singolo pixel nella modalità a 16 colori 720 per 348 della scheda InColor. La scheda InColor non è dotata dell'equivalente funzionale del registro di selezione funzione dell'EGA, quindi questa routine contiene quattro subroutine separate che eseguono operazioni AND, OR o XOR sui valori di pixel.

```
                TITLE   'Listato 48'
                NAME    SetPixelInC
                PAGE    55,132

;
; Nome:         SetPixelInC
;
; Funzione:     Imposta il valore di un pixel nella modalità a 16 colori
;               720x348
;
; Chiamante:    Microsoft C:
;
;                       azzera SetPixel(x,y,n);
```

```
;                       int x,y;            /* coordinate del pixel */
;
;                       int n;              /* valore del pixel */
;

ARGx            EQU     word ptr [bp+4]     ; indirizzamento cornice
                                            ;  di stack
ARGy            EQU     word ptr [bp+6]
ARGn            EQU     byte ptr [bp+8]

DefaultRWColorEQU       0Fh             ; valore di default per reg colore
                                        ;  lettura/scrittura


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddrHGC:near

                PUBLIC  _SetPixelInC
_SetPixelInC    PROC    near

                push    bp              ; mantiene registri chiamante
                mov     bp,sp

                mov     ax,ARGy         ; AX := y
                mov     bx,ARGx             ; BX := x
                call    PixelAddrHGC    ; AH := maschera di bit
                                        ; ES:BX - buffer
                                        ; CL := # bit da spostare
                                        ;  a sinistra

                shl     ah,cl           ; AH := maschera di bit nella
                                        ;  posizione corretta

                mov     dx,3B4h         ; DX := porta CRTC

                jmp     word ptr
                        Set PixelOpInC  ; salta a routine di
                                        ;  sostituzione, AND, OR o XOR


ReplacePixelInC:                        ; routine per sostituire
                                        ;  valore di pixel

                mov     ch,ah           ; CH := maschera di bit per
                                        ;  il pixel
                mov     ax,1F19h        ; AH bit 6 := 0 (polarità
                                        ;  maschera)
                                        ; AH bit 5-4 := 1 (modalità di
                                        ; scrittra)
                                        ; AH bit 3-0 := bit "ininfluenti"
                                        ; AL := numero registro controllo
                                        ; lettura/scrittura
                out     dx,ax           ; imposta reg. controllo
```

```
                                ;  lett./scrit.

            inc     ax          ; AL := 1Ah (numero reg colore
                                ;  lettura/scrittura)
            mov     ah,ARGn     ; AH := valore primo piano
            out     dx,ax       ; imposta reg. colore
                                ;  lettura/scritt.

            and     es:[bx],ch  ; aggiorna piani di bit
            jmp     short L01

ANDPixelInC:                    ; routine per effettuare AND del
                                ; alore del pixel

            mov     ch,ah       ; CH := maschera di bit per
                                ;  il pixel
            mov     ax,1F19h    ; AH bit 6 := 0 (polarità maschera)
                                ; AH bit 5-4 := 1 (modalità di
                                ;  scrittura)
                                ; AH bit 3-0 := bit "ininfluenti"
                                ; AL := numero registro controllo
                                ;  lettura/scrittura
            out     dx,ax       ; imposta reg. controllo
                                ;  lett./scrit.

            dec     ax          ; AL := 18h (numero registro di
                                ;  maschera di piano)
            mov     ah,ARGn     ; AH := valore di pixel
            mov     cl,4
            shl     ah,cl       ; AH bit 7-4 := maschera di piano
                                ;  scrivibile
            or      ah,0Fh      ; AH bit 3-0 := maschera di piano
                                ;  visibile
            out     dx,ax       ; imposta reg. di maschera di piano

            mov     ax,001Ah    ; AH := 0 (valore di primo piano)
                                ; AL := 1Ah (reg. colore let./scrit.)
            out     dx,ax       ; imposta registro colore let./scrit.

            and     es:[bx],ch  ; aggiorna piani di bit
            jmp     short L01


                                ; routine per eseguire OR del valore
                                ;  di pixel
ORPixelInC:
            mov     ch,ah       ; CH := maschera di bit per il pixel
            mov     ax,1F19h    ; AH bit 6 := 0 (polarità maschera)

                                ; AH bit 5-4 := 1 (modalità di
                                ;  scrittura)
                                ; AH bit 3-0 := bit "ininfluenti"
                                ; AL := numero registro controllo
                                ;  lettura/scrittura
            out     dx,ax       ; imposta reg. controllo
                                ;  lett./scrit.
```

```
            dec     ax              ; AL := 18h (numero registro di
                                    ;  maschera di piano)
            mov     ah,ARGn         ; AH := valore di pixel
            not     ah              ; AH := complemento del valore di
                                    ;  pixel
            mov     cl,4
            shl     ah,cl           ; AH bit 7-4 := maschera di piano
                                    ;  scrivibile
            or      ah,0Fh          ; AH bit 3-0 := maschera di piano
                                    ;  visibile
            out     dx,ax           ; imposta reg. di maschera di piano

            mov     ax,0F1Ah        ; AH := 0 (valore di primo piano)
                                    ; AL := 1Ah (reg colore lett./scrit.)
            out     dx,ax           ; imposta reg. colore lettura/scritt.

            and     es:[bx],ch      ; aggiorna piani di bit
            jmp     short L01
XORPixelInC:                        ; routine per eseguire XOR del
                                    ;  valore di pixel
            mov     ch,ah           ; CH := maschera di bit per il pixel
            mov     ax,3F19h        ; AH bit 6 := 0 (polarità maschera)
                                    ; AH bit 5-4 := 3 (modalità di
                                      scrittura)
                                    ; AH bit 3-0 := bit "ininfluenti"
                                    ; AL := numero registro controllo
                                    ;  lettura/scrittura
            out     dx,ax           ; imposta reg. controllo lett./scrit.

            dec     ax              ; AL := 18h (numero registro di
                                    ;  maschera di piano)
            mov     ah,ARGn         ; AH := valore di pixel
            not     ah              ; AH := complemento del valore di
                                    ;  pixel
            mov     cl,4
            shl     ah,cl           ; AH bit 7-4 := maschera di piano
                                    ;  scrivibile
            or      ah,0Fh          ; AH bit 3-0 := maschera di piano
                                    ;  visibile
            out     dx,ax           ; imposta reg. di maschera di piano



            xor     es:[bx],ch      ; aggiorna piani di bit
            jmp     short L01

L01:        mov     ax,0F18h
            out     dx,ax           ; ripristina valori di default
                                    ;  della
                                    ;  maschera di piano

            mov     ax,4019h        ; ripristina valore di default
                                    ;  controllo lettura/scrittura
            out     dx,ax

            inc     ax              ; ripristina valore di default
                                    ;  colore
                                    ;  lettura/scrittura
```

```
                mov     ah,DefaultRWColor
                out     dx,ax

                mov     sp,bp           ; ripristina reg. chiamante
                                        ;  e ritorna
                pop     bp
                ret

_SetPixelInC    ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

SetPixelOpInC   DW      ReplacePixelInc         ; contiene indirizzo
                                                ;  dell'operazione di pixel

_DATA           ENDS

                END
```

Listato 48. *Impostazione di un valore di pixel nella modalità grafica InColor.*

Ognuna di queste subroutine inizia con la programmazione dei registri di controllo lettura/scrittura, colore lettura/scrittura e maschera di piano. Successivamente un'operazione di lettura di CPU carica i latch, ed una successiva operazione di scrittura di CPU aggiorna i piani di bit.

Ogni subroutine inizia con la programmazione del registro di controllo lettura/scrittura di una delle quattro modalità grafiche di scrittura. Contemporaneamente, i bit "ininfluenti" vengono tutti impostati a 1 e il bit di polarità di maschera viene azzerato in modo che il decodificatore ritorni sempre 11111111B come risultato di un'operazione di lettura di CPU. Vengono quindi impostati i valori di primo piano di maschera di piano e di colore lettura/scrittura; questi valori variano a seconda del fatto che il valore del pixel debba essere sostituito o debba essere manipolato con un'operazione AND, OR o XOR.

L'istruzione *AND ES:[BX],CH* (oppure *XOR ES:[BX],CH* per un'operazione XOR del pixel) produce le operazioni di lettura e di scrittura di CPU. Nel corso dell'operazione di lettura, i latch vengono caricati e il valore 11111111B viene ritornato alla CPU; la CPU effettua l'AND (oppure l'XOR) di questo valore con la maschera di bit di CH e riscrive il risultato allo stesso indirizzo del buffer del video. In questo modo, la maschera di bit di CH seleziona quale valore di pixel viene aggiornato durante l'operazione di scrittura di CPU.

Ad eccezione del pixel specificato dalla maschera di bit, il contenuto dei latch viene ricopiato nei piani di bit da cui era stato letto; il valore del pixel che viene aggiornato deriva dal valore di primo piano del registro di colore lettura/scrittura. Solo i piani di bit specificati dal registro di maschera di piano vengono modificati, quindi i soli bit dei piani di bit che vengono aggiornati sono quelli che vengono modificati dalle operazioni di sostituzione, AND, OR o XOR.

CONSIGLIO

E' istruttivo confrontare l'interazione tra i valori di modalità di scrittura, colore di primo piano e maschera di piano all'interno di ogni subroutine. L'operazione logica che ha luogo (sostituzione, AND, OR o XOR) non viene programmata esplicitamente con un'istruzione 80x86. Essa è implicitamente contenuta nei registri di controllo grafico, che sono programmati per emulare l'operazione logica modificando i bit individuali nel pixel aggiornato.

MCGA

Nelle modalità grafiche compatibili CGA, le stesse routine per l'impostazione dei valori di pixel operano senza alcuna modifica sia su CGA sia su MCGA. Le due modalità non CGA (a 2 colori 640 per 480 e a 256 colori 320 per 200) possono essere gestite facilmente con semplici modifiche alla routine per la modalità a 2 colori 640 per 200. I Listati 49 e 50 mostrano le modifiche necessarie.

```
                TITLE   'Listato 49'
                NAME    SetPixel11
                PAGE    55,132

;
; Nome:         SetPixel11
;
; Funzione:     Imposta il valore di un pixel nella modalità a 2 colori
;               640x480 (MCGA o VGA)
;
; Chiamante:    Microsoft C:
;
;                       azzera SetPixel(x,y,n);
;
;                       int x,y;                /* coordinate del pixel */
;
;                       int n;                  /* valore del pixel */
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento
                                                ;  cornice di stack
ARGy            EQU     word ptr [bp+6]

ARGn            EQU     byte ptr [bp+8]


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr10:near
```

```
                PUBLIC _SetPixel11
_SetPixel11     PROC   near

                push   bp              ; mantiene registri del chiamante
                mov    bp,sp

                mov    ax,ARGy         ; AX := y
                mov    bx,ARGx         ; BX := x
                call   PixelAddr10     ; AH := maschera di bit
                                       ; ES:BX -> buffer
                                       ; CL := # bit da spostare a sinistra

                mov    al,ARGn         ; AL := valore di pixel non spostato
                shl    ax,cl           ; AH := maschera di bit nella
                                       ;  posizione corretta
                                       ; AL := valore di pixel nella
                                       ;  posizione corretta

                jmp    word ptr
                Set PixelOp11          ; salta alla routine di
                                       ;  sostituzione,AND, OR o XOR

                                       ; routine per sostituire il
                                       ;  valore di pixel

ReplacePixel11:not     ah              ; AH := inverte maschera di bit
                and    es:[bx],ah      ; azzera il valore del pixel
                or     es:[bx],al      ; imposta il valore del pixel
                jmp    short L02


                                       ; routine per effettuare AND del
                                         valore del pixel
ANDPixel11:     test   al,al
                jnz    L02             ; non effettua nulla se valore di
                                       ;  pixel = 1


L01:            not    ah              ; AH := inversione maschera di bit
                and    es:[bx],ah      ; imposta a 0 bit nel buffer del
                                       ;  video
                jmp    short L02

                                       ; routine per effettuare OR del
                                       ;  valore del pixel
ORPixel11:      test   al,al
                jz     L02             ; non effettua nulla se valore di
                                       ;  pixel = 0

                or     es:[bx],al      ; imposta bit nel buffer del video
                jmp    short L02

                                       ; routine per effettuare XOR del
                                       ;  valore del pixel
XORPixel11:     test   al,al
                jz     L02             ; non effettua nulla se valore di
                                       ;  pixel = 0
```

```
                xor     es:[bx],al      ; effettua XOR del bit nel buffer
                                        ;  del video


02:             mov     sp,bp           ; ripristina reg. chiamante e
                                        ;  ritorna
                pop     bp
                ret

_SetPixel11     ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

SetPixel0p11    DW      ReplacePixel11          ; contiene indirizzo
                                                ;  dell'operazione di pixel

_DATA           ENDS

                END
```

Listato 49. *Impostazione di un valore di pixel nella modalità a 2 colori 640 per 480 dell'MCGA o della VGA.*

```
                TITLE   'Listato 50'
                NAME    SetPixel13
                PAGE    55,132
;
; Nome:         SetPixel13
;
; Funzione:     Imposta il valore di un pixel nella modalità a 256 colori
;               320x200 (MCGA o VGA)

;
; Chiamante:    Microsoft C:
;
;                       azzera SetPixel(x,y,n);
;
;                       int x,y;                /* coordinate del pixel */
;
;                       int n;                  /* valore del pixel */
;

ARGx            EQU     word ptr [bp+4]         ; indirizzamento cornice
                                                ;  di stack
ARGy            EQU     word ptr [bp+6]
ARGn            EQU     byte ptr [bp+8]

DGROUP          GROUP   _DATA


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP
```

```
                EXTRN   PixelAddr13:near

                PUBLIC  _SetPixel13
_SetPixel13     PROC    near

                push    bp              ; mantiene registri del chiamante
                mov     bp,sp

                mov     ax,ARGy         ; AX := y
                mov     bx,ARGx         ; BX := x

                call    PixelAddr13     ; ES:BX - buffer

                mov     al,ARGn         ; AL := valore del pixel

                jmp     word ptr SetPixel Op13
                                        ; salta alla routine di
                                        ;  sostituzione, AND, OR o XOR

ReplacePixel13:         mov         es:[bx],al
                jmp     short L01

ANDPixel13:     and     es:[bx],al
                jmp     short L01

ORPixel13:      or      es:[bx],al
                jmp     short L01

XORPixel13:     xor     es:[bx],al


L01:            mov     sp,bp           ; ripristina reg. chiamante
                                        ;  e ritorna
                pop     bp
                ret



_SetPixel13     ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

SetPixelOp13    DW      ReplacePixel13

_DATA           ENDS

                END
```

Listato 50. *Impostazione di un valore di pixel nella modalità a 256 colori 320 per 200 dell'MCGA o della VGA.*

VGA

Una volta create le routine per l'aggiornamento dei pixel su MCGA e EGA, produrre le stesse routine per VGA è semplice. La sola modalità video della VGA che non esiste sugli altri sottosistemi è la modalità a 16 colori 640 per 480. L'indirizzamento del pixel in questa modalità è identico a quello della modalità a 16 colori 640 per 350 dell'EGA, quindi possono essere utilizzate le routine dei Listati da 45 a 47.

Riempimento del buffer del video

Di solito la prima operazione che si effettua dopo aver selezionato una nuova modalità video è quella di vuotare il buffer del video riempiendolo con uno sfondo uniforme di dati ripetitivi. Nelle modalità alfanumeriche, risulta facile ed efficace riempire il buffer con spazi o caratteri nulli utilizzando l'istruzione *STOSW* dell'80x86.

Il riempimento del buffer del video nelle modalità grafiche è più complicato. L'azzeramento dell'intero buffer è relativamente facile, ma il riempimento dello schermo con un colore pieno o con una configurazione di pixel è più difficile, in particolare su EGA, VGA e su scheda InColor.

CGA

Su CGA, potete impostare l'intero buffer su un singolo valore di pixel o su una configurazione di strisce verticali con un'operazione *REP STOSW*, come viene realizzato dalla routine del Listato 51. Grazie all'intercalazione bidirezionale della mappa di buffer del video, questa tecnica riempie tutte le linee di scansione con numeri pari prima di riempire quelle con numeri dispari. Potreste preferire cancellare il buffer dall'alto al basso riempiendolo una linea per volta. Questa tecnica, utilizzata nel Listato 52, produce un effetto visivo più gradevole, ma richiede un codice più corposo e più lento.

```
            mov     di,0B800h
            mov     es,di
            xor     di,di           ; ES:DI -> inizio del buffer del video
            mov     al,11110000b    ; AL := retinatura di pixel
            mov     ah,al           ; AX := retinatura di pixel duplicata
            mov     cx,2000h        ; CX := numero di parole nel buffer
                                    ;  del video
            rep     stosw           ; riempie il buffer con la retinatura
                                    ;  di pixel

; questo può essere realizzato anche utilizzando il BIOS del video

            mov     ah,0Fh          ; AH := 0Fh (numero di funzione INT 10H)
```

```
        int     10h             ; rileva stato corrente del video;
                                ;  AH = numero delle colonne di carattere
        mov     dl,ah           ; DL := numero di colonne di carattere

        mov     ax,600h         ; AH := 6 (numero di funzione INT 10H)
                                ; AL := 0 (numero di righe da scorrere)
        mov     bh,11110000b    ;  BH := retinatura di pixel
        mov     cx,0            ; CH := 0 (colonna del carattere in
                                ;  alto a sinistra)
                                ; CL := 0 (riga del carattere in alto
                                ;  a sinistra)
        mov     dh,18h          ; DH := 18h (riga del carattere in
                                ;  basso a destra)
        dec     dl              ; DL := colonna del carattere in basso
                                ;  a destra
        int     10h
```

Listato 51. *Semplice riempimento del buffer nella modalità grafica del CGA.*

```
        mov     di,0B800h
        mov     es,di
        xor     di,di           ; ES:DI -> inizio del buffer del video

        mov     al,11001100b    ; AL := retinatura di pixel
        mov     ah,al           ; AX := retinatura di pixel duplicata
        mov     bx,100h         ; CX := numero di coppie di righe

L01:    mov     cx,40           ; CX := numero di parole in ogni riga
        rep     stosw           ; riempie riga pari

        add     di,2000h-80     ; ES:DI -> riga dispari
        mov     cx,40
        rep     stosw           ; riempie riga dispari




        sub     di,2000h        ; ES:DI -> prossima riga pari
        dec     bx
        jnz     L01
```

Listato 52. *Riempimento del buffer nella modalità grafica del CGA utilizzando l'intercalazione a due pagine.*

Potete sfruttare l'intercalazione a due pagine della mappa del buffer del video per creare una nuova sfumatura di colore o un semplice pattern (vedere Listato 53). In questo caso, il pattern di pixel delle linee di scansione con numeri pari viene spostato in posizione dal pattern delle linee di scansione con numeri dispari. Ciò crea sullo schermo una retinatura discontinua o a mezzi-toni. Dal momento che i pixel sono così vicini, l'occhio tende a fonderli, percependo la retinatura discontinua come grigio nella modalità a 2 co-

lori 640 per 200 o come sfumatura di colore intermedio nella modalità a 4 colori 320 per 200.

```
        mov     di,0B800h
        mov     es,di
        xor     di,di           ; ES:DI -> inizio della riga di pixel 0

        mov     al,10101010b    ; AL := retinatura di pixel per
                                ;  righe pari
        mov     ah,al           ; AX := retinatura di pixel duplicata
        mov     cx,1000h        ; CX := numero di parole nel buffer
                                ;  del video
        rep     stosw           ; riempie righe di pixel pari

        mov     di,2000h        ; ES:DI - inizio di riga 1 di pixel
        mov     al,01010101b    ; AL := retinatura di pixel per
                                ;  righe dispari
        mov     ah,al
        mov     cx,1000h
        rep     stosw           ; riempie righe di pixel dispari
```

Listato 53. *Riempimento del buffer nella modalità grafica con diverse configurazioni di pixel nelle righe pari e dispari.*

HGC e HGC+

Potete utilizzare le stesse tecniche di base per la cancellazione del buffer del video sia nella modalità grafica monocromatica 720 per 348 dell'HGC sia nella modalità a 2 colori 640 per 200 del CGA. Tuttavia, la vostra routine deve essere in grado di cancellare entrambe le parti visualizzabili del buffer del video dell'HGC. Il Listato 54 dimostra come è possibile ottenere questo. Anche in questo caso, potete sfruttare la mappa di memoria del video intercalata per creare una retinatura discontinua mentre cancellate il buffer.

```
        mov     es,BufferSeg    ; ES := 0B000h per prima pagina
                                ;  di visualizzazione o 0B800h per
                                ;  seconda pagina di visualizzazione
        xor     di,di           ; ES:DI -> primo byte da riempire

        mov     al,10101010b    ; AL retinatura di pixel
        mov     ah,al           ; AX := retinatura di pixel duplicata

L01:    mov     cx,1000h        ; CX := numero di parole di
                                ;  ogni intercalazione di buffer di 8 KB
        rep     stosw           ; riempie intercalazione; incrementa
                                ;  DI di 2000h

        ror     ax,1            ; sposta retinatura di pixel tra righe
```

```
        or      di,di
        jns     L01             ; salta se DI > 8000h
```

Listato 54. *Riempimento del buffer nella modalità grafica dell'HGC usando l'intercalazione a quattro pagine.*

EGA e VGA

Il controller grafico può fornire una certa quantità di assistenza hardware nel riempimento del buffer del video dell'EGA e della VGA. Inoltre, dal momento che il buffer contiene più dati di quanti se ne possano visualizzare sullo schermo, potete scegliere di cancellare solo la parte visualizzata, una parte non visualizzata o l'intero buffer.

Nelle modalità a 2 colori 640 per 200 e a 4 colori 320 per 200 potete usare le routine per il CGA (vedere Listati da 51 a 53). Ricordate però, che l'EGA e la VGA hanno abbastanza RAM video per supportare due schermi di dati nella modalità a 4 colori 320 per 200. La vostra routine dovrà quindi essere in grado di cancellare qualsiasi area indicata del buffer. Il riempimento del buffer del video nella modalità a 2 colori 640 per 480 (vedere Listato 55) e nella modalità a 256 colori 320 per 200 (vedere Listato 56) risulta anche relativamente semplice, in quanto l'indirizzamento dei pixel in queste modalità è un'operazione facile.

```
        mov     di,0A000h
        mov     es,di
        xor     di,di           ; ES:DI -> inizio del buffer del video
        mov     al,01010101b    ; AL := retinatura di pixel
        mov     ah,al           ; AX := retinatura di pixel duplicata
        mov     cx,480*40       ; CX := (righe di pixel) * (parole
                                ;  per riga)
        rep     stosw           ; riempie il buffer con la retinatura
                                ;  di pixel


; questo può essere realizzato anche utilizzando il BIOS del video

        mov     ax,1130h        ; AH := 11h (numero di funzione INT 10H)
                                ; AL := 30h (informazioni generatore
                                ;  di caratteri)
        int     10h             ; rileva informaz.
                                ; DL = num. di
                                ;  righe di caratt. - 1

        mov     ax,600h         ; AH := 6 (numero di funzione INT 10H)
                                ; AL := 0 (numero di righe da scorrere)
        mov     bh,01010101b    ; BH := retinatura di pixel
        mov     cx,0            ; CH := 0 (colonna del carattere in
                                ;  alto a sinistra)
```

```
                                ; CL := 0 (riga del carattere in alto
                                ;  a sinistra)
        mov     dh,dl           ; DH := (riga del carattere in basso
                                ;  a destra)
        mov     dl,4Fh          ; DL := 4Fh (colonna del carattere
                                ;  in basso a destra)
        int     10h
```

Listato 55. *Riempimento del buffer nella modalità grafica a 2 colori 640 per 480 dell'MCGA e della VGA.*

```
        mov     di,0A000h
        mov     es,di
        xor     di,di           ; ES:DI -> inizio del buffer del video
        mov     ah,PixelValue1  ; AX := configurazione a 2 pixel
        mov     al,PixelValue2
        mov     bx,100          ; BX := numero di coppie di righe

L01:    mov     cx,160          ; CX := numero di parole per riga
        rep     stosw           ; riempie riga con numero pari
        xchg    ah,al           ; scambia pixel di retinatura

        mov     cx,160
        rep     stosw           ; riempie riga con numero dispari
        xchg    ah,al           ; scambia pixel di retinatura

        dec     bx
        jnz     L01
```

Listato 56. *Riempimento del buffer nella modalità grafica a 256 colori 320 per 200 dell'MCGA e della VGA. Questa routine riempie righe di pixel alternate in modo separato per permettere retinature di pixel discontinue.*

Nelle modalità grafiche a 16 colori e 200 linee e in tutte le modalità grafiche a 350 linee, le vostre routine dovrebbero programmare il controller grafico in modo che sfrutti le sue capacità di elaborazione in parallelo. Il metodo più efficace per riempire il buffer del video con un colore pieno è quello di utilizzare la modalità di scrittura 0 per copiare ripetutamente il valore di impostazione/azzeramento nel buffer del video. Dal momento che non è richiesta alcuna operazione di lettura di CPU per questa operazione, potete impostare l'intero buffer del video su un colore pieno con una singola istruzione *REP STOSW* come mostrato nel Listato 57.

```
        mov     di,0A000h
        mov     es,di
        xor     di,di           ; ES:DI -> inizio del buffer del video

        mov     dx,3CEh         ; DX := porta di I/O del controller
                                ;  grafico

        mov     ah,PixelValue   ; AH := valore del pixel per il
```

```
                                ;  riempimento
        mov     al,0            ; AL := 0 (numero registro
                                ;  impostazione/azzeramento)
        out     dx,ax           ; carica registro impostazione/azzeramento

        mov     ax,0F01         ; AH := 1111b (maschera per abilitaz.
                                ;  impostaz./azzer.)
                                ; AL := 1 (numero registro abilitaz.
                                ;  impostaz./azzer.)
        out     dx,ax           ; carica registro abilitaz.
                                ;  impostazione/azzeramento

        mov     cx,PixelRows*40 ; CX := (righe di pixel) *
                                ;  (parole per riga)
        rep     stosw           ; riempie il buffer

        mov     ax,0001         ; AH := 0 (valore di default abilitaz.
                                ;  impost./azzer.)
                                ; AL := 1 (numero registro abilitaz.
                                ;  impostaz./azzer.)
        out     dx,ax           ; ripristina default abilita
                                ;  impostazione/azzeramento
```

Listato 57. *Riempimento con colore pieno del buffer per le modalità grafiche dell'EGA e della VGA. Il codice assume che il controller grafico sia già in modalità di scrittura 0 (default del BIOS).*

Il riempimento del buffer del video con una retinatura di pixel arbitraria è più difficile. Sebbene la tecnica di base sia la stessa, ogni componente della retinatura deve essere scritto separatamente sui piani di bit. L'esempio del Listato 58 riempie il buffer del video con una configurazione di 8 per 2 pixel nella modalità a 16 colori 640 per 480 della VGA. Potete adattare la routine alle modalità a 16 colori a 200 linee e a 350 linee sia su EGA sia su VGA.

```
        mov     di,0A000h
        mov     es,di
        xor     di,di       ; ES:DI -> inizio del buffer del video

        mov     dx,3CEh     ; DX := porta di I/O del controller grafico

        mov     ax,105h     ; AH bit 0-1 := 01b (modalità di scrittura 1)
                            ; AL := 5 (registro modalità grafica)
        out     dx,ax       ; stabilisce modalità di scrittura 1

        mov     ax,PixelRows/2
                            ; AX := numero di coppie di righe
                            ;  da riempire

L04:    mov     cl,es:[0]   ; retinatura pixel di latch per righe pari
        mov     cx,40       ; CX := parole per riga di pixel
        rep     stosw       ; copia i latch su riga a numero pari
```

```
mov     cl,es:[50] ; retinatura pixel di latch
                   ;  per righe dispari
mov     cx,40
rep     stosw      ; riempie riga a numero dispari

dec     ax
jnz     L01        ; loop procedendo verso il basso del buffer

mov     ax,0005
out     dx,ax      ; ripristina modalità di scrittura 0
                   ;   (default)
```

Listato 58. *Riempimento del buffer con retinatura per le modalità grafiche originali dell'EGA e della VGA. Il codice assume che la retinatura di pixel desiderata sia già memorizzata nei primi otto pixel delle prime due righe del buffer del video (cioè a A000:0000 e a A000:0050).*

Scheda InColor

Come con EGA e VGA, dovreste usare i latch di dati grafici della scheda InColor per aggiornare i quattro piani di bit in parallelo. Il riempimento del buffer del video con un colore pieno è semplice, come mostrato nel Listato 59. Il riempimento del buffer con una retinatura di pixel richiede lo stesso tipo di logica applicata nella routine equivalente per EGA e per VGA (mostrata nel Listato 60).

```
mov         es,BufferSeg      ; ES := 0B000h per la prima pagina di
                              ;  visualizzazione o 0B800h per la
                              ;  seconda pagina di visualizzazione
xor         di,di             ; ES:DI -> primo byte da riempire

mov         dx,3B4h           ; DX := porta di I/O del registro
                              ;  di controllo
mov         ah,PixelValue     ; AH := valore del pixel per il riempimento
mov         al,1Ah            ; AL := 1AH (numero registro colore
                              ;  lettura/scrittura)
out         dx,ax             ; carica registro colore lettura/scrittura
mov         ax,4019h          ; AH bit 5-6 := 00b (modalità
                              ;  di scrittura 0)

                              ; AL := 19H (registro controllo
                              ;  lettura/scrittura)
                              ; carica registro controllo
                              ;  lettura/scrittura

            mov     ax,0FFFFh ; AX := maschera di bit del pixel
            mov     cx,4000h  ; CX := numero di parole nel buffer (32K / 2)
            rep     stosw     ; riempie il buffer
```

Listato 59. *Riempimento del buffer con colore pieno per la modalità grafica di InColor Hercules.*

```
        mov     es,Buffer
                Seg         ; ES := 0B000h per prima pagina di
                            ;  visualizzazione o 0B800h per seconda
                            ;  pagina di visualizzaz.
        xor     di,di       ; ES:DI -> primo byte da riempire

        mov     dx,3B4h     ; DX := porta di I/O del registro
                            ;  di controllo

        mov     ax,6019h    ; AH bit 5-6 := 10b (modalità
                            ;  di scrittura 2)
                            ; AL := 19H (registro controllo
                            ;  lettura/scrittura)
        out     dx,ax       ; carica registro controllo
                            ;  lettura/scrittura

        mov     ax,0FFFFh   ; AX := maschera di bit del pixel

L01:    mov     cl,es:[0]   ; retino pixel di latch per righe pari
        mov     cx,1000h    ; CX := numero di parole di
                            ;  ogni intercalazione di buffer di 8 KB
        rep     stosw       ; riempie intercalazione a numero pari;
                            ;  incrementa DI di 2000h

        mov     cl,es:[2000]    ; retino pixel di latch per righe dispari
        mov     cx,1000
        rep     stosw       ; riempie intercalazione a numero dispari

        or      di,di
        jns     L01         ; loop mentre DI < 8000H

        mov     ax,44019h   ; ripristina valore di default di
        out     dx,ax       ;  registro di controllo lettura/scrittura
```

Listato 60. *Riempimento del buffer con retinatura per la scheda InColor. Il codice assume che la retinatura di pixel desiderata sia già memorizzata nei primi otto pixel delle prime due righe del buffer del video (cioè agli offset 0 e 2000H in BufferSeg).*

MCGA

Per riempire il buffer del video su MCGA, nelle modalità grafiche equivalenti, potete utilizzare le routine scritte per il CGA e per la VGA.

6
Linee

Un efficiente algoritmo per il tracciamento delle linee
Conversione a scansione di una linea retta
L'algoritmo di Bresenham

Ottimizzazione
Indirizzamento di pixel efficiente
Confronti di prestazioni - Casi particolari

Implementazioni di PC e PS/2
Routine modulari
Riduzione degli accessi al buffer del video
Calcoli di indirizzo efficienti
CGA - HGC - EGA
MCGA - VGA
Scheda InColor

Attributi di linea

Definizione della linea
Definizione della linea pixel per pixel - Un approccio violento
Un migliore algoritmo

La maggior parte delle applicazioni grafiche si basa su routine che tracciano linee rette sullo schermo. Le linee rette sono componenti di molte immagini grafiche, tra cui i poligoni, le aree riempite di colore (costituite da gruppi di linee contigue) e le curve (costituite da una serie di brevi segmenti di linea uniti alle loro estremità). Dal momento che le linee vengono utilizzate frequentemente nella grafica di visualizzazione, avete bisogno di veloci subroutine di tracciamento linee per ottenere della grafica di visualizzazione con alte prestazioni. Il presente capitolo descrive come costruire delle routine di tracciamento linee efficienti e flessibili per i sottosistemi di visualizzazione IBM.

Un efficiente algoritmo per il tracciamento delle linee

Immaginate cosa accadrebbe se tentaste di tracciare una linea retta dipingendo i quadratini di un foglio a quadretti (vedere Figura 69). Il risultato non sarebbe proprio una linea, ma un gruppo di quadratini che assomigliano ad una linea.

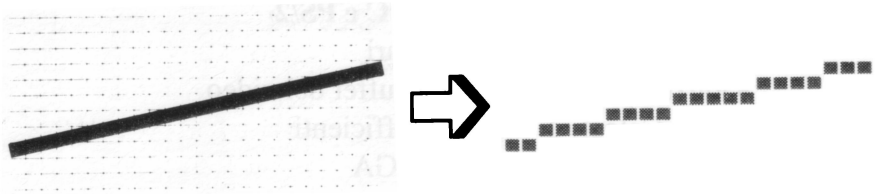


Figura 69. *Linea tracciata su un foglio a quadretti.*

La griglia rettangolare di pixel del quadro dello schermo assomiglia ad un foglio a quadretti "elettronico" quando si tracciano linee rette e curve geometriche. Il masssimo che si può ottenere è la rappresentazione di ogni linea o curva con un gruppo di pixel che producono un effetto che si avvicina il più possibile a quello che si vorrebbe ottenere. Il processo della determinazione di quale serie di pixel del buffer del video assomiglia di più ad una particolare figura geometrica viene definito conversione a scansione.

CONSIGLIO

La conseguenza visiva della conversione a scansione è che le linee e le curve matematicamente continue appaiono discontinue sullo schermo. Considerate la linea quasi orizzontale della Figura 70a. L'unico modo per rappresentare una linea di questo tipo all'interno di un reticolo di pixel è sotto forma di una serie di segmenti di linea orizzontali collegati fra loro. Meno orizzontale o verticale è la linea, più apparirà discontinua. Sebbene delle tecniche software sofisticate possano ridurre l'effetto disconti-

nuo di una linea convertita in scansione, il metodo più semplice per rendere più continua la linea è rappresentato dall'uso di un "reticolo più fitto" e cioè dall'impiego di una modalità di visualizzazione a più alta risoluzione o di un monitor hardware a più alta risoluzione (vedere Figura 70b).

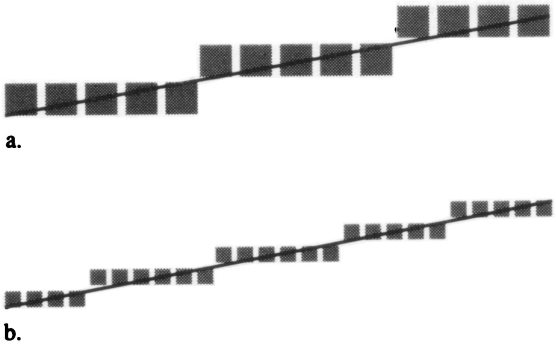


Figura 70. *Una lina quasi orizzontale visualizzata con (a) bassa risoluzione e (b) alta risoluzione.*

Conversione a scansione di una linea retta

Il metodo più semplice per tracciare una linea è quello di utilizzare l'equazione della linea

```
y = mx+b
```

dove *m* è l'inclinazione della linea e *b* è il punto di incrocio *y* (il valore di *y* al punto dove la linea incrocia l'asse *y*). Potete utilizzare questa equazione per calcolare la coordinata *y* corrispondenti per ogni coordinata *x* di pixel tra i punti estremi della linea come mostrato nel Listato 61. Questa tecnica è lenta, ma facile da realizzare.

```
/* Listato 61 */

Line( x1, y1, x2, y2, n )
int           x1,y1;            /* estremità */
int           x2,y2;            /* estremità */
int           n;                /* valore del pixel */
{
              int    x,y;
              float. m;        /* inclinazione */
              float  b;        /* incrocio y */
```

```
            if (x2 == x1)                   /* linea verticale */
            {
             if (y1 > y2)
              Swap( &y1, &y2 );             /* forza y1 > y2 */

             for (y=y1; y <= y2; y++)       /* traccia da y1 a y2 */
              SetPixel( x1, y, n );

             return;
            }

            if (x1 > x2)                    /* forza x 1< x2 */
            {
              Swap( &x1, &x2 );
              Swap( &y1, &y2 );
            }


            m =(float)(y2-y1)/(float)(x2-x1);/* calcola m e b */
            b = y1    (m*x1);

            for (x=x1;x <= 2; x++)          /* traccia da x1 a x2 */
            {
              y = m*x + b;
              SetPixel( x, y, n );
            }
}

Swap( a, b )                                /* scambia valori di a e b */
int         *a,*b;
{
            int    t;

            t = *a;
            *a = *b;
            *b   t;
}
```

Listato 61. *Tracciamento di una linea utilizzando l'equazione della linea.*

Il problema è che le operazioni di servizio di calcolo per eseguire la moltiplicazione, l'addizione e gli arrotondamenti necessari per generare *y* per ogni *x* nella linea sono notevoli. Inoltre, l'inclinazione *m* deve essere mantenuta come numero in virgola mobile, e l'uso dell'aritmetica in virgola mobile nei calcoli li rallenta.

L'algoritmo di Bresenham

Il calcolo incrementale delle corrette coordinate *y* risulta molto più efficiente. Avendo le coordinate *x* e *y* del primo pixel della linea, potete calcolare la locazione di ogni pixel successivo incrementando le coordinate *x* e *y* in proporzione all'inclinazione della linea.

L'aritmetica è più semplice e più veloce di quella necessaria con l'uso diretto dell'equazione della linea.

L'algoritmo presentato da J. E. Bresenham nel 1965 (IBM System Journal 4 (1) 1965, pagg. 25-30) traccia la serie di pixel che si trovano più vicino alla linea tracciata tra due dati pixel ($x1,y1$) e ($x2,y2$), assumendo che $x1$ sia minore di $x2$ e che l'inclinazione della linea sia tra 0 e 1. Per semplificare l'equazione della linea, l'algoritmo assume che la locazione della prima estremità ($x1,y1$) sia (0,0). L'equazione della linea risultante è

```
y=(dy/dx)*x
```

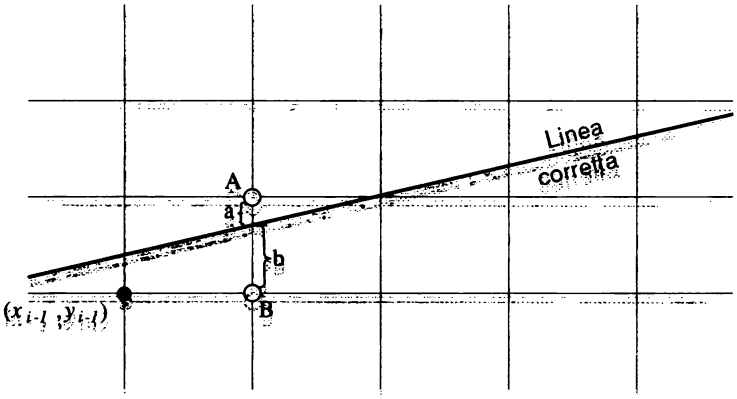
dove

```
dy=y2-y1
```

e

```
dx=x2-x1
```

Per visualizzare come funziona l'algoritmo di Bresenham, considerate la parte di linea mostrata nella Figura 63. L'algoritmo procede determinando iterativamente la coordinata y corrispondente per ogni valore di x da $x1$ a $x2$. Dopo aver tracciato il pixel a (x_{i-1},y_{i-1}), ad esempio, l'algoritmo determina se il pixel A o il pixel B è più vicino alla linea esatta e traccia il pixel più vicino.




```
Avendo (xi-1,yi-1);
se (a<b), (xi,yi)@pixelA
altrimenti (xi,yi)@pixelB
```

Figura 71. *Algoritmo di Bresenham di tracciamento linea incrementale. Avendo il pixel a (x_{i-1},y_{i-1}), l'algoritmo seleziona o il pixel A o il pixel B a seconda dei valori di a e b.*

La differenza tra la coordinata *y* del pixel *A* e la coordinata *y* sulla linea esatta a x_i è

$$a=(y_i+1)-(dy/dx)*xi_I$$

dove (*dy/dx*) rappresenta l'inclinazione della linea. Analogamente, la distanza *b* tra il pixel *B* e la linea è

$$b=(dy/dx)*x_i-y_i$$

Se la distanza *b* è inferiore alla distanza *a*, il pixel *B* si trova più vicino alla linea corretta. Se *a* è inferiore a *b*, il pixel *A* si trova più vicino alla linea corretta. In altre parole, il segno della differenza (*b-a*) determina se è il pixel *A* o il pixel *B* a trovarsi più vicino alla linea.

A questo punto, tutte queste operazioni potrebbero sembrare molto più complesse del semplice utilizzo dell'equazione della linea. Tuttavia, i valori di *a* e di *b* possono essere confrontati implicitamente per ogni x_i, calcolando iterativamente il valore di (*b-a*) per ogni x_i successiva, in termini di quantità più semplici come *dy* e *dx*. Il calcolo risultante è semplice, sebbene ricavarlo richieda un minimo d'algebra.

Per ricavare il calcolo, combinate le equazioni di *a* e di *b*:

$$(b-a) = 2*(dy/dx)*x_i-2*y_i-1$$

Dal momento che *xl* è inferiore a *x2*, *dx* è sempre positivo, quindi *dx*(b-a)* può essere utilizzato al posto di *(b-a)* per decidere se tracciare il pixel *A* o il pixel *B*:

$$idx*(b-a) = 2*dy*x_i-2*y_i*dx-dx$$

$$= 2*(dy*x_i-dx*y_i)-dx$$

Supponiamo che di rappresenti la quantità dx*(b-a). Per calcolare di iterativamente, dovete sapere come calcolarlo da di-1:

$$(di-d_{i-1}) = (2*(dy*x_i-dx*y_i))-(2*(dy*x_{i-1}-dx*y_{i-1}))$$

$$=2*(dy*(x_i-x_{i-1})-dx*(y_i-y_{i-1}))$$

x_i-x_{i-1} è sempre 1, e y_i-y_{i-1} è 1 (se viene tracciato il pixel *A* a (x_i,y_i+*1*)) o 0 (se viene tracciato il pixel *B* a (x_i,y_i)). Quindi, il calcolo della differenza tra d_i e d_{i-1} è semplice, e d_i può essere calcolato semplicemente incrementando d_{i-1} con una delle due costanti possibili:

Se d_{i-1}>=0, traccia pixel *A* a (x_i,y_i+*1*). L'incremento per d_{i-1} è quindi

$$(d_i-d_{i-1}) = 2*(dy-dx)$$

Se *d*i-1 < 0, traccia pixel *B* a (x_i,y_i). L'incremento per d_{i-1} è quindi

$$(d_i-d_{i-1}) = 2*dy$$

Per calcolare il valore iniziale di d_i, ricordate che si presume che il primo pixel della linea è a (0,0). Sostituendo d_i con x_i=1 e y_i=0 nell'equazione otteniamo

$$d_i = 2*dy-dx$$

L'algoritmo risultante è efficace, dal momento che i calcoli più complicati vengono effettuati solo una volta, al di fuori del loop che traccia i pixel (vedere Listato 62). All'interno del loop, la determinazione incrementale di quali pixel si trovano più vicini alla linea desiderata (utilizzando la variabile decisionale d_i) elimina la necessità di utilizzare l'aritmetica in virgola mobile che risulta più lenta. Quello che si ottiene è un velocissimo algoritmo di tracciamento linea.

Ottimizzazione

Ciononostante, c'è ancora spazio per un miglioramento. La parte più lenta dell'abbozzo di routine di tracciamento linea del Listato 62 è la chiamata a *SetPixel()*, che calcola l'indirizzo di pixel nel buffer del video e che quindi imposta il valore del pixel. Il calcolo dell'indirizzo del pixel è chiaramente la parte più lenta della procedura.

```
/* Listato 62 */

Line( x1, y1, x2, y2, n )        /* per linee con inclinazione tra -1 e 1 */
int           x1,y1;
int           x2,y2;             /* estremità */
int           n;                 /* valore di pixel */
{
              int    d,dx,dy;
              int    Aincr,Bincr,yincr;
              int    x,y;


              if (x1> x2)                  /* forza x1< x2 */
              {
                Swap( &x1, &x2 );
                Swap( &y1, &y2 );
              }

              if (y2> y1)                  /* determina incremento per y */
                yincr = 1;
              else
                yincr = -1;

              dx = x2 - x1;                /* inizializza costanti */
              dy = abs( y2-y1 );
              d = 2 * dy - dx;

              Aincr = 2 * (dy - dx);
              Bincr = 2 * dy;
```

```
            x = x1;                     /* x e y iniziali */
            y = y1;

            SetPixel( x, y, n );        /* imposta pixel a (x1,y1) */

            for (x=x1+1; x<=x2; x++)    /* procede da x1+1 a x2 */
            {
              if (d> = 0)
              {
                y += yincr;             /* imposta pixel A */
                d += Aincr;
              }
              else                      /* imposta pixel B */
                d += Bincr;

              SetPixel( x, y, n );
            }
}


Swap( pa, pb )
int         *pa,*pb;
{
            int    t;

            t = *pa;
            *pa = *pb;
            *pb = t;
}
```

Listato 62. *Un' implementazione ad alto livello dell' algoritmo di Bresenham.*

Indirizzamento di pixel efficiente

Fortunatamente, potete ottimizzare in modo significativo il calcolo dell'indirizzamento del pixel: gli indirizzi stessi del pixel possono essere calcolati in modo incrementale, allo stesso modo in cui si incrementa la variabile decisionale d_i. Dopo aver calcolato l'indirizzo del primo pixel della linea, potete trovare i pixel adiacenti nel buffer del video o incrementando l'offset di byte del pixel o ruotando la maschera di bit che rappresenta il relativo offset di bit. Il calcolo degli indirizzi di pixel in modo incrementale è significativamente più veloce dell'esecuzione del calcolo a partire da zero per ogni coppia di coordinate (x,y) della linea.

Ad esempio, potete identificare immediatamente il pixel alla destra di un dato pixel ruotando di una posizione di pixel verso destra la maschera di bit del pixel conosciuto (se il pixel conosciuto è il pixel all'estrema destra del byte, incrementate anche l'offset del byte). Per ricavare il pixel immediatamente superiore ad un pixel conosciuto, decrementate l'offset del byte per il numero di byte per riga di pixel, ma tenete invariata la maschera di bit. Questo calcolo è leggermente più complicato nelle modalità video con mappa di buffer di video intercalata, ma il principio è identico.

Confronti di prestazioni

Quando confrontate le tecniche per la conversione a scansione delle linee, i miglioramenti di prestazioni derivanti dall'uso dei calcoli di indirizzo incrementali e dell'algoritmo di tracciamento linea incrementale sono notevoli (vedere Figura 72). Anche la scrittura delle vostre routine di tracciamento linee utilizzando il linguaggio Assembler può portare ad un ulteriore miglioramento. La codifica e l'ottimizzazione della manipolazione della maschera di bit e dei calcoli degli indirizzi risultano molto più semplici in linguaggio Assembler che non in un linguaggio ad alto livello.

Algoritmo	Linguaggio	Pixel al secondo
Algoritmo basato sull'equazione della linea	C	4.800
Algoritmo di Bresenham	C	16.000
Algoritmo di Bresenham	Assembler	26.000
Algoritmo di Bresenham con calcolo dell'indirizzo di pixel incrementale	Assembler	70.000

Figura 72. *Prestazioni degli algoritmi di tracciamento linee in C e in linguaggio Assembler. I tempi indicati sono stati ottenuti con un PC/AT IBM a 6 MHz dotato di scheda grafica Hercules.*

Casi particolari

Per migliorare ulteriormente le prestazioni complessive dei driver grafici di visualizzazione, utilizzate le speciali routine per il tracciamento delle linee orizzontali e verticali. In molte applicazioni, questi casi particolari rientrano in una sorprendente percentuale di chiamate alla routine di base di tracciamento linee. Questo è particolarmente vero se utilizzate linee per riempire delle aree.

```
/*-Listato 63 */

FilledRectangle( x1, y1, x2, y2, n )
int             x1,y1;                  /* angolo superiore sinistro */
int             x2,y2;                  /* angolo inferiore destro */
int             n;                      /* valore di pixel */
{
                int     y;

                for (y=y1; y <= y2; y++)     /* traccia rettangolo come*/
                    Line( x1, y, x2, y, n ); /*  serie di linee */
                                             /*  orizzontali adiacenti
}
```

Listato 63. *Una routine che traccia linee orizzontali.*

Ad esempio, la routine *FilledRectangle()* del Listato 63 richiama la funzione di tracciamento linee per tracciare esclusivamente linee orizzontali. Se riempite un rettangolo che misura 100 pixel di altezza, la funzione di tracciamento linee viene richiamata 100 volte per tracciare una linea orizzontale. Quando la funzione di tracciamento linee riconosce il caso particolare delle linee orizzontali, le funzioni come *FilledRectangle()* agiscono molto più velocemente.

Una routine apposita può tracciare linee orizzontali 10 volte più velocemente di una routine di tracciamento linee generica. Per le linee verticali, una routine apposita risulta più veloce di circa il 25 percento. Le linee orizzontali vengono rappresentate nel buffer del video da sequenze contigue di byte che potete riempire con un'istruzione *REP STOSB* dell'80x86, questa istruzione opera molto più velocemente del loop iterativo richiesto dalla routine di base per il tracciamento linee. Nel tracciamento di linee verticali, non è richiesta della logica per determinare le locazioni di pixel. Si incrementa semplicemente l'indirizzo di pixel. Anche in questo caso, il codice risultante è più semplice e più veloce.

Implementazioni di PC e PS/2

Le implementazioni dell'algoritmo di Bresenham per il tracciamento delle linee su hardware di visualizzazione IBM sono fortemente influenzate dalle capacità della CPU e dalle idiosincrasie della mappatura di pixel all'interno del buffer del video nelle varie modalità grafiche. Ciononostante, una volta scritta una routine di tracciamento linee per una modalità grafica, potete adattare il codice sorgente ad altre modalità grafiche o ad altri dispositivi hardware di visualizzazione con poche difficoltà.

Routine modulari

Dovreste costruire le vostre routine di tracciamento linee con una struttura modulare. Un metodo pratico per dividere il vostro codice in moduli è quello di scrivere routine separate per le linee orizzontali, le linee verticali, le linee con inclinazione inferiore a 1 e le linee con inclinazione superiore a 1. Ogni modulo in sé comprende una serie di moduli per l'esecuzione di ogni manipolazione di pixel necessaria (XOR, AND, OR e sostituzione).

CONSIGLIO

L'algoritmo di Bresenham, come si può dedurre da questo capitolo, è applicabile solamente alle linee le cui inclinazioni variano da 0 a 1. Tuttavia, è semplice utilizzare lo stesso algoritmo per linee con altre inclinazioni. Per le linee con inclinazioni tra 1 e 0, è sufficiente modificare il segno dell'incremento *y* (vedere Listato 62). Per le linee con inclinazioni inferiori a -1 o maggio-

ri di 1 (cioè, $|(dy/dx)|>1$, utilizzate lo stesso algoritmo ma modificate le coordinate x e y.
Ad esempio, ognuna delle routine di tracciamento linee del linguaggio Assembler di questo capitolo contiene due subroutine analoghe, una per $|(dy/dx)| < 1$ ed un'altra per $|(dy/dx)|$. Ogni routine contiene un preambolo che rileva i casi particolari delle linee orizzontali e verticali, inizializza i valori di incremento appropriati e seleziona la corretta subroutine per l'inclinazione.

La suddivisione delle routine in moduli serve soprattutto quando si personalizza il codice per un'applicazione particolare ed anche a semplificare il compito della scrittura del codice in modo che operi simmetricamente nelle diverse modalità grafiche. Ad esempio, una routine che traccia una linea verticale nella modalità a 2 colori 640 per 200 su CGA richiede una leggera modifica per poter operare correttamente nella modalità a 4 colori 320 per 200.

Riduzione degli accessi al buffer del video

Nella famiglia di microprocessori 8086, le istruzioni di trasferimento dati del formato *MOV mem,reg* sono tra le più lente. Tentate di ridurre al minimo indispensabile l'uso di questa istruzione di CPU all'interno delle vostre routine di base di tracciamento linee. I pixel adiacenti di una linea sono frequentemente raggruppati nello stesso byte del buffer del video (ovviamente, tali gruppi si presentano più frequentemente nelle linee che più si avvicinano ad essere orizzontali). Potete accelerare le vostre routine di tracciamento linee aggiornando tutti i pixel adiacenti di ogni byte memorizzato nel buffer del video.

Calcoli di indirizzo efficienti

Per ottimizzare le prestazioni, usate in modo accurato i registri di CPU per memorizzare i valori più frequentemente aggiornati nei loop interni delle vostre routine: la maschera di bit di pixel, l'offset del buffer e la variabile decisionale. Nel Listato 64, ad esempio, i registri DH e DL contengono le maschere di bit, il registro BX contiene l'offset di byte e la variabile decisionale *d* viene memorizzata nel registro DI. Questi valori sono quelli aggiornati più frequentemente all'interno di queste routine, quindi sono quelli che dovreste tentare di tenere nei registri piuttosto che nelle variabili di memoria.

Se trascurate l'uso efficace dei registri di CPU, le vostre routine potrebbero operare molto più lentamente del necessario. Considerate cosa accadrebbe se riscriveste la routine del Listato 64 per memorizzare la variabile decisionale in una variabile di memoria invece che nel registro DI. Solo questo cambiamento secondario provocherebbe un rallentamento di circa il 20 percento della routine (questo non solo enfatizza la ragione per cui dovreste utilizzare al meglio i registri di CPU, ma fa anche intuire perché la scrittura di routine grafiche di visualizzazione ottimizzate in un linguaggio ad alto livello sia molto difficile).

CGA

Il Listato 64 contiene il codice per il tracciamento di linee nella modalità grafica a 2 colori 640 per 200 del CGA. La routine è costituita da un preambolo e da quattro moduli di tracciamento linee. Il preambolo dispone le estremità in ordine ascendente in base alle loro coordinate *x*, imposta gli appropriati incrementi verticali per calcolare l'indirizzo di pixel nel loop interno e seleziona un appropriato modulo di tracciamento linee sulla base dell'inclinazione della linea. I moduli di tracciamento linee (*VertLine06*,*HorizLine06*,*LoSlopeLine06*, e *HiSlopeLine06*) contengono i loop interni che aggiornano i pixel ed incrementano gli indirizzi.

```
                TITLE   'Listato 64'
                NAME    Line06
                PAGE    55,132

;
; Nome:         Line06
;
; Funzione:     Traccia una linea nella modalità a 2 colori 640x200
;
; Chiamante:    Microsoft C:
;
;                       azzera Line06(x1,y1,x2,y2,n);
;
;                       int x1,y1,x2,y2;        /* coordinate del pixel */
;
;                       int n;                  /* valore di pixel */
;

ARGx1           EQU     word ptr [bp+4]   indirizzamento cornice di stack
ARGy1           EQU     word ptr [bp+6]
ARGx2           EQU     word ptr [bp+8]
ARGy2           EQU     word ptr [bp+10]
ARGn            EQU     byte ptr [bp+12]

VARleafincr     EQU     word ptr [bp-2]
VARincr1        EQU     word ptr [bp-4]
VARincr2        EQU     word ptr [bp-6]
VARroutine      EQU     word ptr [bp-8]

ByteOffsetShift EQU     3                 usato per convertire i pixel in
                                           offset di byte

DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr06:

                PUBLIC  _Line06
```

```
_Line06         PROC    near

                push    bp              ; mantiene registri del chiamante
                mov     bp,sp
                sub     sp,8            ; spazio di stack per le variabili
                                        ;  locali
                push    si
                push    di


                mov     si,2000h        ; incremento per intercalazione di
                                        ;  buffer del video
                mov     di,80-2000h     ; incremento dall'ultima alla prima
                                        ;  intercalazione

                mov     cx,ARGx2
                sub     cx,ARGx1        ; CX := x2 - x1
                jz      VertLine06      ; salta se linea verticale

; forza x1 < x2

                jns     L01             ; salta se x2> x1

                neg     cx              ; CX := x1 - x2

                mov     bx,ARGx2        ; scambia x1 e x2
                xchg    bx,ARGx1
                mov     ARGx2,bx

                mov     bx,ARGy2        ; scambia y1 e y2
                xchg    bx,ARGy1
                mov     ARGy2,bx

; calcola dy = ABS(y2-y1)

L01:            mov     bx,ARGy2
                sub     bx,ARGy1        ; BX := y2 - y1
                jnz     L02

                jmp     HorizLine06     ; salta se linea orizzontale

L02:            jns     L03

                neg     bx              ; BX := y1 - y2
                neg     si              ; nega incrementi per interc. buffer
                neg     di
                xchg    si,di           ; scambia incrementi

; seleziona routine appropriata per inclinazione linea

L03:            mov     VARleafincr,di  ; salva incremento per interc. buffer

                mov     VARroutine,offset LoSlopeLine06
                cmp     bx,cx
                jle     L04             ; salta se dy <= dx (inclinaz.<= 1)
                mov     VARroutine,offset HiSlopeLine06
```

```
            xchg    bx,cx           ; scambia dy e dx

; calcola variabile decisionale iniziale e incrementa

L04:        shl     bx,1            ; BX := 2 * dy
            mov     VARincr1,bx     ; incr1 := 2 * dy
            sub     bx,cx
            mov     di,bx           ; DI := d = 2 * dy - dx
            sub     bx,cx
            mov     VARincr2,bx     ; incr2 := 2 * (dy - dx)

; calcola primo indirizzo di pixel

            push    cx              ; mantiene questo registro
            mov     ax,ARGy1        ; AX := y
            mov     bx,ARGx1        ; BX := x
            call    PixelAddr06     ; AH := maschera di bit
                                    ; ES:BX -> buffer
                                    ; CL := # bit da spostare a sinistra

            mov     al,ARGn         ; AL := valore di pixel non spostato
            shl     ax,cl           ; AH := maschera di bit nella
                                    ;  posizione corretta
                                    ; AL := valore di pixel nella
                                    ;  posizione corretta

            mov     dx,ax           ; DH := maschera di bit
                                    ; DL := valore di pixel
            not     dh              ; DH := inverte maschera di bit

            pop     cx              ; ripristina questo registro
            inc     cx              ; CX := # di pixel da tracciare

            test    bx,2000h        ; imposta flag zero se BX è nella
                                    ;  prima intercalazione
            jz      L05

            xchg    si,VARleafincr  ; scambia valori di incremento se il
                                    ;  primo pixel si trova nella prima
                                    ;  intercalazione


L05:        jmp     VARroutine      ; salta a routine appropriata per
                                    ;  inclinazione


; routine per linee verticali

VertLine06: mov     ax,ARGy1        ; AX := y1
            mov     bx,ARGy2        ; BX := y2
            mov     cx,bx
            sub     cx,ax           ; CX := dy
            jge     L31             ; salta se dy> = 0

            neg     cx              ; forza dy> = 0
            mov     ax,bx           ; AX := y2
```

```
L31:        inc     cx              ; CX := # di pixel da tracciare
            mov     bx,ARGx1        ; BX := x
            push    cx              ; mantiene questo registro
            call    PixelAddr06     ; AH := maschera di bit
                                    ; ES:BX -> buffer del video
                                    ; CL := # bit da spostare a sinistra
            mov     al,ARGn         ; AL := valore di pixel
            shl     ax,cl           ; AH := maschera di bit nella
                                    ;  posizione corretta
                                    ; AL := valore di pixel nella
                                    ;  posizione corretta
            not     ah              ; AH := inverte maschera di bit
            pop     cx              ; ripristina questo registro

            test    bx,si           ; imposta flag zero se BX è nella
                                    ;  prima intercalazione
            jz      L32

            xchg    si,di           ; scambia valori di incremento se il
                                    ;  primo pixel si trova nella prima
                                    ;  intercalazione

L32:        test    al,al
            jz      L34             ; salta se valore di pixel = 0

L33:        or      es:[bx],al      ; imposta valori di pixel nel buffer
            add     bx,si           ; incrementa all'area successiva di
                                    ;  intercalazione
            xchg    si,di           ; commuta tra valori di incremento
            loop    L33             ; loop verso il basso della linea
            jmp     short L35

L34:        and     es:[bx],ah      ; azzera valori di pixel nel buffer
            add     bx,si           ; incrementa all'area successiva
                                    ;  dell'intercalazione
            xchg    si,di           ; commuta tra valori di incremento
            loop    L34

L35:        jmp     Lexit

; routine per linee orizzontali (inclinazione = 0)

HorizLine06: mov    ax,ARGy1
            mov     bx,ARGx1
            call    PixelAddr06;    ; AH := maschera di bit
                                    ; ES:BX -> buffer del video
                                    ; CL := # bit da spostare a sinistra
            mov     di,bx           ; ES:DI -> buffer

            mov     dh,ah
            not     dh              ; DH := maschera di bit non spostata
                                    ;       per byte all'estrema sinistra
            mov     dl,0FFh         ; DL := maschera di bit non spostata
                                    ;       per byte all'estrema destra
```

```
            shl     dh,cl           ; DH := inverte maschera di bit per
                                    ;  primo byte
            not     dh              ; DH := maschera di bit per primo
                                    ;  byte

            mov     cx,ARGx2
            and     cl,7
            xor     cl,7            ; CL := numero di bit da spostare a
                                    ;  sinistra
            shl     dl,cl           ; DL := maschera di bit per ultimo
                                    ;  byte

; determina offset byte del primo e dell'ultimo pixel della linea

            mov     ax,ARGx2        ; AX := x2
            mov     bx,ARGx1        ; BX := x1

            mov     cl,ByteOffsetShift   ; numero di bit da spostare per
                                         ;  convertire i pixel in byte

            shr     ax,cl           ; AX := offset di byte di x2
            shr     bx,cl           ; BX := offset di byte di x1
            mov     cx,ax
            sub     cx,bx           ; CX := (# byte nella linea) - 1

; estende valore di pixel per tutto un byte

            mov     bx,offset DGROUP:PropagatedPixel
            mov     al,ARGn         ; AL := valore di pixel
            xlat

; imposta pixel nel byte all'estrema sinistra della linea

            or      dh,dh
            js      L43             ; salta se è allineato al byte (x1 è
                                    ;  il pixel all'estrema sinistra nel
                                    ;  byte)
            or      cx,cx
            jnz     L42             ; salta se c'è più di un byte nella
                                    ;  linea

            and     dl,dh           ; maschera di bit per la linea
            jmp     short L44

L42:        mov     ah,al
            and     ah,dh           ; AH := bit mascherati di pixel
            not     dh              ; DH := inverte maschera di bit per
                                    ;  primo byte
            and     es:[di],dh      ; azzera pixel mascherati nel buffer
            or      es:[di],ah      ; aggiorna pixel mascherati nel
                                    ;  buffer
            inc     di
            dec     cx

; usa una veloce istruzione macchina 8086 per tracciare il resto della
;linea
```

```
L43:            rep     stosb           ; aggiorna tutti i pixel nella linea

; imposta pixel nel byte all'estrema destra della linea

L44:            and     al,dl           ; AL := pixel mascherati per ultimo
                                        ;  byte
                not     dl
                and     es:[di],dl      ; azzera pixel mascherati nel buffer
                or      es:[di],al      ; aggiorna pixel mascherati nel
                                        ;  buffer

                jmp     Lexit


; routine per dy<= dx (inclinazione<= 1)   ; ES:BX -> buffer del video
                                        ; CX = # pixel da tracciare
                                        ; DH = inverte maschera di bit
                                        ; DL = valore di pixel nella posizione
                                        ;  corretta
                                        ; SI = incremento dell'intercalazione
                                        ;  del buffer
                                        ; DI = variabile decisionale
LoSlopeLine06:

L10:            mov     ah,es:[bx]      ; AH := byte del buffer del video

L11:            and     ah,dh           ; azzera valore di pixel all'offset
                                        ;  corrente di bit
                or      ah,dl           ; imposta valore di pixel nel byte

                ror     dl,1            ; ruota valore di pixel
                ror     dh,1            ; ruota maschera di bit
                jnc     L14             ; salta se maschera di bit è stata
                                        ; ruotata sulla posizione di pixel
                                        ; all'estrema sinistra

; maschera di bit non spostata fuori

                or      di,di           ; test del segno di d
                jns     L12             ; salta se d => 0

                add     di,VARincr1     ; d := d + incr1
                loop    L11

                mov     es:[bx],ah      ; memorizza restanti pixels nel
                                        ;  buffer
                jmp     short Lexit

L12:            add     di,VARincr2;    ; d := d + incr2
                mov     es:[bx],ah      ; aggiorna buffer

                add     bx,si           ; incrementa y
                xchg    si,VARleafincr  ; scambia valori di incremento
                                        ;  intercalazione
                loop    L10
                jmp     short Lexit
```

```
; maschera di bit spostata fuori

L14:            mov     es:[bx],ah      ; aggiorna buffer
                inc     bx              ; BX := offset del byte successivo

                or      di,di           ; test del segno di d
                jns     L15             ; salta se non negativo

                add     di,VARincr1     ;d := d + incr1
                loop    L10
                jmp     short Lexit

L15:            add     di,VARincr2     ; d := d + incr2

                add     bx,si           ; incrementa y
                xchg    si,VARleafincr

                loop    L10
                jmp     short Lexit


; routine per dy> dx (inclinazione> 1)     ; ES:BX -> buffer del video
                                        ; CX = # pixel da tracciare
                                        ; DH = inverte maschera di bit
                                        ; DL = valore di pixel nella
                                        ;  posizione corretta
                                        ; SI = incremento dell'intercalazione
                                        ;  del buffer
                                        ; DI = variabile decisionale
HiSlopeLine06:

L21:            and     es:[bx],dh      ; azzera valore di pixel nel buffer
                                        ;  del video
                or      es:[bx],dl      ; imposta valore di pixel nel byte

                add     bx,si           ; incrementa y
                xchg    si,VARleafincr  ; scambia valori di incremento
                                        ;  intercalazione


L22:            or      di,di           ; test del segno di d
                jns     L23             ; salta se d> = 0

                add     di,VARincr1     ; d := d + incr1
                loop    L21

                jmp     short Lexit


L23:            add     di,VARincr2     ; d := d + incr2

                ror     dl,1            ; ruota valore di pixel
                ror     dh,1            ; ruota maschera di bit
                cmc                     ; cf impostato se maschera di bit non
                                        ;  è stata ruotata su posizione di
                                        ;  pixel all'estrema sinistra
```

```
                adc     bx,0            ; BX := offset del byte successivo

                loop    L21


Lexit:          pop     di              ; ripristina registri e ritorna
                pop     si
                mov     sp,bp
                pop     bp
                ret

_Line06         ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

PropagatedPixel DB      00000000b   ; 0
                DB      11111111b   ; 1

_DATA             ENDS

                  END
```

Listato 64. *Una routine di tracciamento linee per la modalità a 2 colori 640 per 200 del CGA.*

La maggior parte del tempo di esecuzione richiesto da questa routine viene occupato dai loop interni dei quattro moduli di tracciamento linee. Per ottimizzare la velocità dei loop interni, più calcoli possibili vengono effettuati all'esterno di essi. In particolare, il loop interno di *HorizLine06* (alla label *L43*) è molto veloce in quanto è costituito solo da una singola istruzione in linguaggio macchina 80x86.

Le routine *LoSlopeLine06* e *HiSlopeLine06* implementano l'algoritmo di Bresenham. Il loop interno di *HiSlopeLine06* (a *L21*) è più semplice del loop interno di *LoSlopeLine06* (a *L11*). Questo perché *HiSlopeLine06* incrementa la coordinata *y* di pixel, e quindi l'offset del buffer, in ogni iterazione, quindi l'unico altro codice necessario nel loop è il codice per incrementare la variabile decisionale e per aggiornare la maschera di bit del pixel. In *LoSlopeLine06*, la coordinata *x* viene incrementata ad ogni iterazione tramite la rotazione della maschera di bit di pixel. Ciò richiede del codice supplementare per l'aggiornamento della maschera di bit e dell'offset del buffer in conformità al valore della variabile decisionale.

La routine per la modalità a 4 colori 320 per 200, mostrata nel Listato 65, è simile a quella per la modalità a 2 colori 640 per 200. In effetti, potreste scrivere una singola routine che operi in entrambe le modalità senza inutili sacrifici di prestazioni. Le differenze risiedono in come viene calcolato l'indirizzo del primo pixel della linea (cioè, tra la chiamata a *PixelAddr04* e quella a *PixelAddr06*) e in quanti bit vengono mascherati e aggiornati per ogni pixel del buffer. La maschera di bit ha la dimensione di 1 bit nella modalità a 2 colori 640 per 200 e di 2 bit nella modalità a 4 colori 320 per 200.

```
                TITLE   'Listato 65'
                NAME    Line04
                PAGE    55,132

;
; Nome:         Line04
;
; Funzione:     Traccia una linea nella modalità a 4 colori 320x200
;
; Chiamante:    Microsoft C:
;
;                       azzera Line04(x1,y1,x2,y2,n);
;
;                       int x1,y1,x2,y2;            /* coordinate del pixel */
;
;                       int n;                      /* valore di pixel */
;

ARGx1           EQU     word ptr [bp+4] ; indirizzamento cornice di stack
ARGy1           EQU     word ptr [bp+6]
ARGx2           EQU     word ptr [bp+8]
ARGy2           EQU     word ptr [bp+10]
ARGn            EQU     byte ptr [bp+12]
VARleafincr     EQU     word ptr [bp-2]
VARincr1        EQU     word ptr [bp-4]
VARincr2        EQU     word ptr [bp-6]
VARroutine      EQU     word ptr [bp-8]

ByteOffsetShift EQU     2               ; usato per convertire i pixel in
                                        ;  offset di byte

DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr04:near

                PUBLIC  _Line06
_Line04         PROC    near

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                sub     sp,8        ; spazio di stack per le variabili locali
                push    si
                push    di


                mov     si,2000h    ; incremento per intercalazione di
                                    ;  buffer del video
                mov     di,80-2000h ; incremento dall'ultima alla prima
                                    ;  intercalazione

                mov     cx,ARGx2
                sub     cx,ARGx1    ; CX := x2 - x1
                jz      VertLine04  ; salta se linea verticale
```

```
; forza x1< x2

            jns     L01         ; salta se x2 > x1

            neg     cx          ; CX := x1 - x2

            mov     bx,ARGx2    ; scambia x1 e x2
            xchg    bx,ARGx1
            mov     ARGx2,bx

            mov     bx,ARGy2    ; scambia y1 e y2
            xchg    bx,ARGy1
            mov     ARGy2,bx

; calcola dy = ABS(y2-y1)

L01:        mov     bx,ARGy2
            sub     bx,ARGy1    ; BX := y2 - y1
            jnz     L02

            jmp     HorizLine04;salta se linea orizzontale

L02:        jns     L03

            neg     bx          ; BX := y1 - y2
            neg     si          ; nega incrementi per intercalazione
                                ;  buffer
            neg     di
            xchg    si,di       ; scambia incrementi

; seleziona routine appropriata per inclinazione linea

L03:        mov     VARleafincr,di ; salva incremento per intercalazione
                                   ;  buffer

            mov     VARroutine,offset LoSlopeLine04
            cmp     bx,cx
            jle     L04         ; salta se dy <= dx (inclinaz. <= 1)
            mov     VARroutine,offset HiSlopeLine04
            xchg    bx,cx       ; scambia dy e dx

; calcola variabile decisionale iniziale e incrementa

L04:        shl     bx,1        ; BX := 2 * dy
            mov     VARincr1,bx; incr1 := 2 * dy
            sub     bx,cx
            mov     di,bx       ; DI := d = 2 * dy - dx
            sub     bx,cx
            mov     VARincr2,bx; incr2 := 2 * (dy - dx)

; calcola primo indirizzo di pixel

            push    cx          ; mantiene questo registro
            mov     ax,ARGy1    ; AX := y
            mov     bx,ARGx1    ; BX := x
            call    PixelAddr04; AH := maschera di bit
                                ; ES:BX -> buffer
```

```
                                ; CL := # bit da spostare a sinistra

            mov     al,ARGn     ; AL := valore di pixel non spostato
            shl     ax,cl       ; AH := maschera di bit nella
                                ;  posizione corretta
                                ; AL := valore di pixel nella
                                ;  posizione corretta

            mov     dx,ax       ; DH := maschera di bit
                                ; DL := valore di pixel
            not     dh          ; DH := inverte maschera di bit

            pop     cx          ; ripristina questo registro
            inc     cx          ; CX := # di pixel da tracciare

            test    bx,2000h    ; imposta flag zero se BX è nella
                                ;  prima intercalazione
            jz      L05

            xchg    si,VARl eafincr; scambia valori di incremento se il
                                   ;  primo pixel si trova nella prima
                                   ;  intercalazione

L05:        jmp     VARroutine  ; salta a routine appropriata per
                                ;  inclinazione


; routine per linee verticali

VertLine04: mov     ax,ARGy1    ; AX := y1
            mov     bx,ARGy2    ; BX := y2
            mov     cx,bx
            sub     cx,ax       ; CX := dy
            jge     L31         ; salta se dy> = 0

            neg     cx          ; forza dy> = 0
            mov     ax,bx       ; AX := y2

L31:        inc     cx          ; CX := # di pixel da tracciare
            mov     bx,ARGx1    ; BX := x
            push    cx          ; mantiene questo registro
            call    PixelAddr04; AH := maschera di bit
                                ; ES:BX - >buffer del video
                                ; CL := # bit da spostare a sinistra
            mov     al,ARGn     ; AL := valore di pixel
            shl     ax,cl       ; AH := maschera di bit nella
                                ;  posizione corretta
                                ; AL := valore di pixel nella
                                ;  posizione corretta
            not     ah          ; AH := inverte maschera di bit
            pop     cx          ; ripristina questo registro

            test    bx,si       ; imposta flag zero se BX è nella
                                ;  prima intercalazione
            jz      L32
```

```
            xchg    si,di       ; scambia valori di incremento se
                                ;  primo pixel si trova nella prima
                                ;  intercalazione

L32:        and     es:[bx],ah  ; azzera pixel nel buffer
            or      es:[bx]al   ; imposta valore di pixel nel buffer

            add     bx,si       ; incrementa all'area successiva di
                                ;  intercalazione
            xchg    si,di       ; commuta tra valori di incremento

            loop    L32
            jmp     Lexit

; routine per linee orizzontali (inclinazione = 0)

HorizLine04:mov     ax,ARGy1
            mov     bx,ARGx1
            call    PixelAddr04; AH := maschera di bit
                                ; ES:BX -> buffer del video
                                ; CL := # bit da spostare a sinistra
            mov     di,bx       ; ES:DI -> buffer

            mov     dh,ah
            not     dh          ; DH := maschera di bit non spostata
                                ;       per byte all'estrema sinistra
            mov     dl,0FFh     ; DL := maschera di bit non spostata
                                ;       per byte all'estrema destra

            shl     dh,cl       ; DH := inverte maschera di bit per
                                ;  primo byte
            not     dh          ; DH := maschera di bit per primo
                                ;  byte

            mov     cx,ARGx2
            and     cl,3
            xor     cl,3
            shl     cl,1        ; CL := numero di bit da spostare a
                                ;  sinistra
            shl     dl,cl       ; DL := maschera di bit per ultimo
                                ;  byte

; determina offset byte del primo e dell'ultimo pixel della linea

            mov     ax,ARGx2    ; AX := x2
            mov     bx,ARGx1    ; BX := x1

            mov     cl,ByteOffsetShift    ; numero di bit da spostare per
                                          ;  convertire i pixel in byte

            shr     ax,cl       ; AX := offset di byte di x2
            shr     bx,cl       ; BX := offset di byte di x1
            mov     cx,ax
            sub     cx,bx       ; CX := (# byte nella linea) - 1

; estende valore di pixel per tutto un byte

            mov     bx,offset DGROUP:PropagatedPixel
```

```
                mov     al,ARGn     ; AL := valore di pixel
                xlat                ; AL := valore di pixel esteso

; imposta pixel nel byte all'estrema sinistra della linea

                or      dh,dh
                js      L43         ; salta se è allineato al byte (x1 è
                                    ;  il pixel all'estrema sinistra nel
                                    ;  byte)
                or      cx,cx
                jnz     L42         ; salta se c'è più di un byte nella
                                    ;  linea

                and     dl,dh       ; maschera di bit per la linea
                jmp     short L44

L42:            mov     ah,al
                and     ah,dh       ; AH := bit mascherati di pixel
                not     dh          ; DH := inverte maschera di bit per
                                    ;  primo byte
                and     es:[di],dh  ; azzera pixel mascherati nel buffer
                or      es:[di],ah  ; aggiorna pixel mascherati nel
                                    ;  buffer
                inc     di
                dec     cx

; usa una veloce istruzione macchina 8086 per tracciare il resto della
; linea

L43:            rep     stosb       ; aggiorna tutti i pixel nella linea

; imposta pixel nel byte all'estrema destra della linea

L44:            and     al,dl       ; AL := pixel mascherati per ultimo
                                    ;  byte
                not     dl
                and     es:[di],dl  ; azzera pixel mascherati nel buffer
                or      es:[di],al  ; aggiorna pixel mascherati nel
                                    ;  buffer

                jmp     Lexit


; routine per dy <= dx (inclinazione <= 1) ; ES:BX -> buffer del video
                                    ; CX = # pixel da tracciare
                                    ; DH = inverte maschera di bit
                                    ; DL = valore di pixel nella
                                    ;  posizione corretta
                                    ; SI = incremento dell'intercalazione
                                    ;  del buffer
                                    ; DI = variabile decisionale
LoSlopeLine04:

L10:            mov     ah,es:[bx]  ; AH := byte del buffer del video
```

```
L11:            and     ah,dh       ; azzera valore di pixel all'offset
                                    ;  corrente di  bit
                or      ah,dl       ; imposta valore di pixel nel byte

                ror     dl,1        ; ruota valore di pixel
                ror     dl,1
                ror     dh,1        ; ruota maschera di bit
                ror     dh,1
                jnc     L14         ; salta se maschera di bit è stata
                                    ;  ruotata sulla posizione di pixel
                                    ;  all'estrema sinistra

; maschera di bit non spostata fuori

                or      di,di       ; test del segno di d
                jns     L12         ; salta se d>= 0

                add     di,VARincr1; d := d + incr1
                loop    L11

                mov     es:[bx],ah ; memorizza restanti pixel nel buffer
                jmp     short Lexit

L12:            add     di,VARincr2; d := d + incr2
                mov     es:[bx],ah ; aggiorna buffer

                add     bx,si       ; incrementa y
                xchg    si,VARleafincr ; scambia valori di incremento
                                    ;  intercalazione

                loop    L10
                jmp     short Lexit

; maschera di bit spostata fuori

L14:            mov     es:[bx],ah ; aggiorna buffer
                inc     bx          ; BX := offset del byte successivo

                or      di,di       ; test del segno di d
                jns     L15         ; salta se non negativo

                add     di,VARincr1; d := d + incr1
                loop    L10
                jmp     short Lexit

L15:            add     di,VARincr2; d := d + incr2

                add     bx,si       ; incrementa y
                xchg    si,VARleafincr

                loop    L10
                jmp     short Lexit


; routine per dy>dx (inclinazione>1)        ; ES:BX -> buffer del video
                                            ; CX = # pixel da tracciare
```

```
                                ; DH = inverte maschera di bit
                                ; DL = valore di pixel nella
                                ;  posizione corretta
                                ; SI = incremento dell'intercalazione
                                ;  del buffer
                                ; DI = variabile decisionale
HiSlopeLine04:

L21:            and     es:[bx],dh ; azzera valore di pixel nel buffer
                                ;  del video
                or      es:[bx],dl ; imposta valore di pixel nel byte

                add     bx,si      ; incrementa y
                xchg    si,VAR1 eafincr; scambia valori di incremento
                                     ;  intercalazione

L22:            or      di,di      ; test del segno di d
                jns     L23        ; salta se d = >0

                add     di,VARincr1; d := d + incr1
                loop    L21

                jmp     short Lexit


L23:            add     di,VARincr2; d := d + incr2

                ror     dl,1       ; ruota valore di pixel
                ror     dl,1
                ror     dh,1       ; ruota maschera di bit
                ror     dh,1
                cmc                ; cf impostato se maschera di bit non
                                   ;  è stata ruotata su posizione di
                                   ;  pixel all'estrema sinistra

                adc     bx,0       ; BX := offset del byte successivo

                loop    L21


Lexit:          pop     di         ; ripristina registri e ritorna
                pop     si
                mov     sp,bp
                pop     bp
                ret

_Line04         ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'
```

```
PropagatedPixel        DB          00000000b ; 0
                       DB          01010101b ; 1
                       DB          10101010b ; 2
                       DB          11111111b ; 3

_DATA                  ENDS

                       END
```

Listato 65. *Una routine di tracciamento linee per la modalità a 4 colori 320 per 200 del CGA.*

Sul CGA il codice che gestisce gli incrementi verticali viene reso complicato dalla necessità di attraversare le intercalazioni del buffer del video. L'indirizzo di pixel viene incrementato di 2000H per passare dalla prima intercalazione (coordinate *y* pari) alla seconda intercalazione (coordinate *y* dispari). Per incrementare da un pixel ad una coordinata *y* dispari al pixel immediatamente sottostante, si aggiunge -2000H (per incrementare dalla seconda alla prima intercalazione) più 80 (il numero di byte di ogni riga di pixel nel buffer). L'incremento è quindi 0E050H (80-2000H).

CONSIGLIO

Le routine per CGA presentate nei Listati 64 e 65 possono solo copiare nel buffer del video il valore di pixel specificato. Per eseguire un'operazione XOR, OR o AND sui valori preesistenti nel buffer utilizzando il valore di pixel specificato, modificate i loop interni di ognuno dei quattro moduli di tracciamento linee. La selezione di una delle operazioni di pixel (XOR, AND, eccetera) pone di fronte al solito compromesso tra la velocità e la dimensione del codice. Per ottimizzare la velocità, scrivete un modulo di tracciamento linee separato per ogni operazione di pixel (AND, OR, XOR e sostituzione). Per ridurre il codice ridondante, richiamate una breve subroutine o aggiungete dei collegamenti logici per eseguire una delle operazioni sui pixel.

HGC

La routine per HGC, contenuta nel Listato 66 è analoga a quella per la modalità a 2 colori 640 per 200 del CGA. La differenza principale è costituita dal modo in cui il buffer del video dell'HGC viene mappato. A causa dell'intercalazione a quattro pagine del buffer del video Hercules, l'indirizzo di pixel viene incrementato aggiungendo il valore di intercalazione di buffer (2000H o -2000H) fino a che il risultato non supera il limite di offset validi di buffer. Dal momento che gli offset validi di buffer variano tra 0 e 7FFFH, la routine rileva la condizione di superamento esaminando il bit più significativo del risultato. Quando il risultato eccede il limite, la routine aggiunge un altro valore (90-8000H o 8000H-90) al risultato, in modo che il nuovo risultato corrisponda all'offset corretto dell'intercalazione di buffer successiva.

```
                TITLE   'Listato 66'
                NAME    LineHGC
                PAGE    55,132

;
; Nome:         LineHGC
;
; Funzione:     Traccia una linea nelle modalità grafiche 720x348 HGC o HGC+
;
; Chiamante:    Microsoft C:
;
;                       azzera LineHGC (x1,y1,x2,y2,n);
;
;                       int x1,y1,x2,y2;            /* coordinate del pixel */
;
;                       int n;                      /* valore di pixel */
;

ARGx1           EQU     word ptr [bp+4] ; indirizzamento cornice di stack
ARGy1           EQU     word ptr [bp+6]
ARGx2           EQU     word ptr [bp+8]
ARGy2           EQU     word ptr [bp+10]
ARGn            EQU     byte ptr [bp+12]
VARleafincr     EQU     word ptr [bp-2]
VARincr1        EQU     word ptr [bp-4]
VARincr2        EQU     word ptr [bp-6]
VARroutine      EQU     word ptr [bp-8]

ByteOffsetShift EQU     3           ; usato per convertire i pixel in offset
                                    ;  di byte

DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddrHGC:near

                PUBLIC  _LineHGC
_LineHGC        PROC    near

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                sub     sp,8        ; spazio di stack per le variabili locali
                push    si
                push    di


                mov     si,2000h    ; incremento per intercalazione di buffer
                                    ;  del video
                mov     di,90-8000h ; incremento dall'ultima alla prima
                                    ;  intercalazione

                mov     cx,ARGx2
                sub     cx,ARGx1    ; CX := x2 - x1
                jz      VertLineHGC ; salta se linea verticale
```

```
; forza x1< x2

            jns     L01         ; salta se x2 > x1

            neg     cx          ; CX := x1 - x2

            mov     bx,ARGx2    ; scambia x1 e x2
            xchg    bx,ARGx1
            mov     ARGx2,bx

            mov     bx,ARGy2    ; scambia y1 e y2
            xchg    bx,ARGy1
            mov     ARGy2,bx

; calcola dy = ABS(y2-y1)

L01:        mov     bx,ARGy2
            sub     bx,ARGy1    ; BX := y2 - y1
            jnz     L02

            jmp     Horiz
                    LineHGC     ; salta se linea orizzontale

L02:        jns     L03

            neg     bx          ; BX := y1 - y2
            neg     si          ; nega incrementi per intercalazione
                                ;  buffer
            neg     di

; seleziona routine appropriata per inclinazione linea

L03:        mov     VARleafincr,di ; salva incremento per intercalazione
                                   ;  buffer

            mov     VARroutine,offset LoSlopeLineHGC
            cmp     bx,cx
            jle     L04         ; salta se dy= < dx (inclinazione <= 1)
            mov     VARroutine,offset HiSlopeLineHGC
            xchg    bx,cx       ; scambia dy e dx

; calcola variabile decisionale iniziale e incrementa

L04:        shl     bx,1        ; BX := 2 * dy
            mov     VARincr1,bx; incr1 := 2 * dy
            sub     bx,cx
            mov     di,bx       ; DI := d = 2 * dy - dx
            sub     bx,cx
            mov     VARincr2,bx; incr2 := 2 * (dy - dx)

; calcola primo indirizzo di pixel

            push    cx          ; mantiene questo registro
            mov     ax,ARGy1    ; AX := y
            mov     bx,ARGx1    ; BX := x
            call    PixelAddrHGC   ; AH := maschera di bit
                                   ; ES:BX -> buffer
```

```
                                ; CL := # bit da spostare a sinistra

            mov     al,ARGn     ; AL := valore di pixel non spostato
            shl     ax,cl       ; AH := maschera di bit nella posizione
                                ;  corretta
                                ; AL := valore di pixel nella posizione
                                ;  corretta

            mov     dx,ax       ; DH := maschera di bit
                                ; DL := valore di pixel
            not     dh          ; DH := inverte maschera di bit

            pop     cx          ; ripristina questo registro
            inc     cx          ; CX := # di pixel da tracciare

            jmp     VARroutine  ; salta a routine appropriata per
                                ;  inclinazione

; routine per linee verticali

VertLineHGC:mov     ax,ARGy1    ; AX := y1
            mov     bx,ARGy2    ; BX := y2
            mov     cx,bx
            sub     cx,ax       ; CX := dy
            jge     L31         ; salta se dy >= 0

            neg     cx          ; forza dy >= 0
            mov     ax,bx       ; AX := y2

L31:        inc     cx          ; CX := # di pixel da tracciare
            mov     bx,ARGx1    ; BX := x
            push    cx          ; mantiene questo registro
            call    Pixel AddrHGC  ; AH := maschera di bit
                                ; ES:BX -> buffer del video
                                ; CL := # bit da spostare a sinistra
            mov     al,ARGn     ; AL := valore di pixel
            shl     ax,cl       ; AH := maschera di bit nella posizione
                                ;  corretta
                                ; AL := valore di pixel nella posizione
                                ;  corretta
            not     ah          ; AH := inverte maschera di bit
            pop     cx          ; ripristina questo registro

; traccia la linea
            test    al,al
            jz      L34         ; salta se valore di pixel è zero

L32:        or      es:[bx],al  ; imposta valori di pixel nel buffer

            add     bx,si       ; incrementa all'area successiva di
                                ;  intercalazione
            jns     L33
            add     bx,di       ; incrementa alla prima area
                                ;  d'intercalazione
L33:        loop    L32
            jmp     short L36
```

```
L34:            and     es:[bx],ah ; azzera valori di pixel nel buffer
                add     bx,si      ; incrementa all'area successiva
                                   ;  dell'intercalazione
                jns     L35
                add     bx,di      ; incrementa alla prima area
                                   ;  d'intercalazione
L35             loop    L34

L36:            jmp     Lexit


; routine per linee orizzontali (inclinazione = 0)

HorizLineHGC:   mov     ax,ARGy1
                mov     bx,ARGx1
                call    Pixel AddrHGC   ; AH := maschera di bit
                                   ; ES:BX -> buffer del video
                                   ; CL := # bit da spostare a sinistra
                mov     di,bx      ; ES:DI -> buffer

                mov     dh,ah
                not     dh         ; DH := maschera di bit non spostata per
                                   ;       byte all'estrema sinistra
                mov     dl,0FFh    ; DL := maschera di bit non spostata per
                                   ;       byte all'estrema destra

                shl     dh,cl      ; DH := inverte maschera di bit per primo
                                   ;  byte
                not     dh         ; DH := maschera di bit per primo byte

                mov     cx,ARGx2
                and     cl,7
                xor     cl,7       ; CL := numero di bit da spostare a
                                   ;  sinistra
                shl     dl,cl      ; DL := maschera di bit per ultimo byte

; determina offset byte del primo e dell'ultimo pixel della linea

                mov     ax,ARGx2   ; AX := x2
                mov     bx,ARGx1   ; BX := x1

                mov     cl,Byte OffsetShift  ; numero di bit da spostare per
                                             ;  convertire i pixel in byte

                shr     ax,cl      ; AX := offset di byte di x2
                shr     bx,cl      ; BX := offset di byte di x1
                mov     cx,ax
                sub     cx,bx      ; CX := (# byte nella linea) - 1

; estende valore di pixel per tutto un byte

                mov     bx,offset DGROUP:PropagatedPixel
                mov     al,ARGn    ; AL := valore di pixel
                xlat               ; AL := estende valore di pixel
```

```
; imposta pixel nel byte all'estrema sinistra della linea

            or      dh,dh
            js      L43         ; salta se è allineato al byte (x1 è il
                                ;   pixel all'estrema sinistra nel byte)
            or      cx,cx
            jnz     L42         ; salta se c'è più di un byte nella linea

            and     dl,dh       ; maschera di bit per la linea
            jmp     short L44

L42:        mov     ah,al
            and     ah,dh       ; AH := bit mascherati di pixel
            not     dh          ; DH := inverte maschera di bit per primo
                                ;  byte
            and     es:[di],dh  ; azzera pixel mascherati nel buffer
            or      es:[di],ah  ; aggiorna pixel mascherati nel buffer
            inc     di
            dec     cx

; usa una veloce istruzione macchina 8086 per tracciare il resto della
; linea

L43:        rep     stosb       ; aggiorna tutti i pixel nella linea

; imposta pixel nel byte all'estrema destra della linea

L44:        and     al,dl       ; AL := pixel mascherati per ultimo byte
            not     dl
            and     es:[di],dl  ; azzera pixel mascherati nel buffer
            or      es:[di],al  ; aggiorna pixel mascherati nel buffer

            jmp     Lexit


; routine per dy <= dx (inclinazione <= 1) ; ES:BX -> buffer del video
                                ; CX = # pixel da tracciare
                                ; DH = inverte maschera di bit
                                ; DL = valore di pixel nella posizione
                                ;  corretta
                                ; SI = incremento dell'intercalazione del
                                ;  buffer
                                ; DI = variabile decisionale
LoSlopeLineHGC:

L10:        mov     ah,es:[bx]  ; AH := byte del buffer del video

L11:        and     ah,dh       ; azzera valore di pixel all'offset
                                ;  corrente di bit
            or      ah,dl       ; imposta valore di pixel nel byte

            ror     dl,1        ; ruota valore di pixel
            ror     dh,1        ; ruota maschera di bit
            jnc     L14         ; salta se maschera di bit è stata
                                ;   ruotata sulla posizione di pixel
                                ;   all'estrema sinistra
```

```
; maschera di bit non spostata fuori

            or      di,di       ; test del segno di d
            jns     L12         ; salta se d> = 0

            add     di,VARincr1 ; d := d + incr1
            loop    L11

            mov     es:[bx],ah  ; memorizza restanti pixel nel buffer
            jmp     short Lexit

L12:        add     di,VARincr2 ; d := d + incr2
            mov     es:[bx],ah  ; aggiorna buffer

            add     bx,si       ; incrementa y
            jns     L13         ; salta se non si trova nell'ultima
                                ;  intercalazione

            add     bx,VARleafincr  ; incrementa nell'intercalazione
                                ;  successiva

L13:        loop    L10
            jmp     short Lexit

; maschera di bit spostata fuori

L14:        mov     es:[bx],ah  ; aggiorna buffer
            inc     bx          ; BX := offset del byte successivo

            or      di,di       ; test del segno di d
            jns     L15         ; salta se non negativo

            add     di,VARincr1 ; d := d + incr1
            loop    L10
            jmp     short Lexit

L15:        add     di,VARincr2 ; d := d + incr2

            add     bx,si       ; incrementa y
            jns     L16         ; salta se non si trova nell'ultima
                                ;  intercalazione

            add     bx,VARleafincr
                                ; incrementa nell'intercalazione
                                ;  successiva

L16:        loop    L10         ; loop fino a che tutti i pixel sono
                                ;  impostati
            jmp     short Lexit


; routine per dy> dx
             (inclinazione > 1)    ; ES:BX -> buffer del video
                                ; CX = # pixel da tracciare
                                ; DH = inverte maschera di bit
```

```
                                ; DL = valore di pixel nella posizione
                                ;  corretta
                                ; SI = incremento dell'intercalazione del
                                ;  buffer
                                ; DI = variabile decisionale
HiSlopeLineHGC:

L21:            and     es:[bx],dh ; azzera valore di pixel nel buffer del
                                ;  video
                or      es:[bx],dl ; imposta valore di pixel nel byte

                add     bx,si      ; incrementa y
                jns     L22        ; salta se non si trova nell'ultima
                                   ;  intercalazione

                add     bx,VARleafincr ; incrementa nell'intercalazione
                                       ;  successiva

L22:            or      di,di
                jns     L23        ; salto se d> = 0

                add     di,VARincr1; d := d + incr1
                loop    L21
                jmp     short Lexit

L23:            add     di,VARincr2; d := d + incr2

                ror     dl,1       ; ruota valore di pixel
                ror     dh,1       ; ruota maschera di bit
                cmc                ; cf impostato se maschera di bit non è
                                   ;  stata ruotata su posizione di pixel
                                   ;  all'estrema sinistra

                adc     bx,0       ; BX := offset del byte successivo

                loop    L21

Lexit:          pop     di
                pop     si
                mov     sp,bp
                pop     bp
                ret

_LineHGC        ENDP

_TEXT           ENDS

_DATA           SEGMENTword public 'DATA'

PropagatedPixel DB   00000000b  ; 0
                DB   11111111b  ; 1

_DATA             ENDS

                  END
```

Listato 66. *Una routine di tracciamento linee per la modalità grafica monocromatica Hercules.*

Le routine per HGC non accedono mai al buffer del video con operazioni di lettura/scrittura a 16 bit come MOVSW o AND [BX],DX. L'evitare queste operazioni a 16 bit assicura che le routine opereranno anche sulla scheda InColor oltre che su HGC e HGC+.

CONSIGLIO

Potete utilizzare le stesse routine di tracciamento linee su entrambe le pagine di visualizzazione di HGC impostando il valore appropriato per *VideoBufferSeg* in *PixelAddrHGC*. Per la pagina di visualizzazione 0, impostate a B000H *VideoBufferSeg*. Per la pagina di visualizzazione 1, usate B800H.

EGA

Per EGA, tre routine di tracciamento linee sono sufficienti a rispondere alle esigenze di tutte le modalità grafiche disponibili. Le routine per le modalità a 2 colori 640 per 200 e a 4 colori 320 per 200 del CGA operano ugualmente bene sulle modalità equivalenti dell'EGA. La routine per le restanti modalità grafiche (le modalità a 16 colori a 200 linee e tutte le modalità a 350 linee) viene complicata dalla necessità di programmare il controller grafico, ma nel contempo viene semplificata dal fatto che l'hardware del controller grafico gestisce una parte delle manipolazioni di pixel che devono essere eseguite in ambito software sul CGA.

La routine del Listato 67 usa la modalità di scrittura 0 del controller grafico per aggiornare il buffer del video. La routine memorizza il valore di pixel per la linea nel registro di impostazione/azzeramento. Per ogni pixel aggiornato nel buffer, la routine scrive la maschera di bit appropriata sul registro di maschera di bit. Di conseguenza, una singola istruzione 80x86 può leggere, aggiornare e riscrivere fino a 8 pixel contemporaneamente.

```
                TITLE   'Listato 67'
                NAME    Line10
                PAGE    55,132

;
; Nome:         Line10
;
; Funzione:     Traccia una linea nelle seguenti modalità grafiche EGA e VGA:
;                       modalità a 16 colori 200 linee
;                       modalità a 350 linee
;                       modalità a 16 colori 640x480
;
; Chiamante:    Microsoft C:
;
;                       azzera Line10(x1,y1,x2,y2,n);
;
```

```
;                       int x1,y1,x2,y2;            /* coordinate del pixel */
;
;                       int n;                      /* valore di pixel */
;

ARGx1           EQU     word ptr [bp+4] ; indirizzamento cornice di stack
ARGy1           EQU     word ptr [bp+6]
ARGx2           EQU     word ptr [bp+8]
ARGy2           EQU     word ptr [bp+10]
ARGn            EQU     byte ptr [bp+12]
VARvertincr     EQU     word ptr [bp-2]
VARincr1        EQU     word ptr [bp-4]
VARincr2        EQU     word ptr [bp-6]
VARroutine      EQU     word ptr [bp-8]

ByteOffsetShift         EQU 3       ; usato per convertire i pixel in
                                    ;  offset di byte
BytesPerLine    EQU     80
RMWbits         EQU     0           ; valore per reg rotazione
                                    ;  dati/selezione funzione


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr10:near

                PUBLIC  _Line10
_Line10         PROC    near

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                sub     sp,8        ; spazio di stack per le variabili
                                    ;  locali
                push    si
                push    di

; configura il controller grafico

                mov     dx,3CEh     ; DX := indirizzo di porta controller
                                    ;  grafico

                mov     ah,ARGn     ; AH := valore di pixel
                xor     al,al       ; AL := numero reg impostaz./azzeram.
                out     dx,ax

                mov     ax,0F01h    ; AH := 1111b (maschera di piano di
                                    ;  bit per abilitaz. impostaz./azzer.
                out     dx,ax       ; AL := # reg abilit. impost./azzer.

                mov     ah,RMWbits  ; bit 3 e 4 di AH := funzione
                mov     al,3        ; AL := # reg rotazione
                                    ;  dati/selezione funzione
                out     dx,ax

; verifica linea verticale
```

```
            mov     si,BytesPerLine; incrementa per buffer video

            mov     cx,ARGx2
            sub     cx,ARGx1    ; CX := x2 - x1
            jz      VertLine10  ; salta se linea verticale

; forza x1 < x2

            jns     L01         ; salta se x2> x1

            neg     cx          ; CX := x1 - x2

            mov     bx,ARGx2    ; scambia x1 e x2
            xchg    bx,ARGx1
            mov     ARGx2,bx

            mov     bx,ARGy2    ; scambia y1 e y2
            xchg    bx,ARGy1
            mov     ARGy2,bx

; calcola dy = ABS(y2-y1)

L01:        mov     bx,ARGy2
            sub     bx,ARGy1    ; BX := y2 - y1
            jz      HorizLine10; salta se linea orizzontale

            jns     L03         ; salta se inclinazione è positiva

            neg     bx          ; BX := y1 - y2
            neg     si          ; nega incremento per interc. buffer

; seleziona routine appropriata per inclinazione linea

L03:        mov     VARleafincr,si ; salva incremento verticale

            mov     VARroutine,offset LoSlopeLine10
            cmp     bx,cx
            jle     L04         ; salta se dy <= dx (inclinaz. < =1)
            mov     VARroutine,offset HiSlopeLine10
            xchg    bx,cx       ; scambia dy e dx

; calcola variabile decisionale iniziale e incrementa

L04:        shl     bx,1        ; BX := 2 * dy
            mov     VARincr1,bx; incr1 := 2 * dy
            sub     bx,cx
            mov     si,bx       ; SI := d = 2 * dy - dx
            sub     bx,cx
            mov     VARincr2,bx; incr2 := 2 * (dy - dx)

; calcola primo indirizzo di pixel

            push    cx          ; mantiene questo registro
            mov     ax,ARGy1    ; AX := y
```

```
            mov     bx,ARGx1    ; BX := x
            call    PixelAddr10; AH := maschera di bit
                                ; ES:BX -> buffer
                                ; CL := # bit da spostare a sinistra

            mov     di,bx       ; ES:DI -< buffer
            shl     ah,cl       ; AH := maschera di bit nella
                                ;  posizione corretta
            mov     bl,ah       ; AH,BL := maschera di bit
            mov     al,8        ; AL := numero reg. maschera di bit

            pop     cx          ; ripristina questo registro
            inc     cx          ; CX := # di pixel da tracciare

            jmp     VARroutine  ; salta a routine appropriata per
                                ;  inclinazione


; routine per linee verticali

VertLine10:  mov    ax,ARGy1    ; AX := y1
             mov    bx,ARGy2    ; BX := y2
             mov    cx,bx
             sub    cx,ax       ; CX := dy
             jge    L31         ; salta se dy> = 0

             neg    cx          ; forza dy> = 0
             mov    ax,bx       ; AX := y2

L31:         inc    cx          ; CX := # di pixel da tracciare
             mov    bx,ARGx1    ; BX := x
             push   cx          ; mantiene questo registro
             call   PixelAddr04; AH := maschera di bit
                                ; ES:BX -> buffer del video
                                ; CL := # bit da spostare a sinistra
; imposta controller grafico

             shl    ah,cl       ; AH := maschera di bit nella
                                ;  posizione corretta
             mov    al,8        ; AL := numero reg maschera di bit
             out    dx,ax

             pop    cx          ; ripristina questo registro

; traccia la linea

L32:         or     es:[bx],al  ; imposta il pixel
             add    bx,si       ; incrementa a linea successiva
             loop   L32

             jmp    Lexit


; routine per linee orizzontali (inclinazione = 0)

HorizLine10:

             push   ds          ; mantiene DS
```

```
            mov     ax,ARGy1
            mov     bx,ARGx1
            call    PixelAddr10; AH := maschera di bit
                               ; ES:BX - buffer del video
                               ; CL := # bit da spostare a sinistra
            mov     di,bx      ; ES:DI - buffer

            mov     dh,ah      ; DH := maschera di bit non spostata
                               ;       per byte all'estrema sinistra
            not     dh
            shl     dh,cl      ; DH := inverte maschera di bit per
                               ;  primo byte
            not     dh         ; DH := maschera di bit per primo
                               ;  byte

            mov     cx,ARGx2
            and     cl,7
            xor     cl,7       ; CL := numero di bit da spostare a
                               ;  sinistra
            mov     dl,0FFh    ; DL := maschera di bit non spostata
                               ;       per byte all'estrema destra
            shl     dl,cl      ; DL := maschera di bit per ultimo
                               ;  byte

; determina offset byte del primo e dell'ultimo pixel della linea

            mov     ax,ARGx2   ; AX := x2
            mov     bx,ARGx1   ; BX := x1

            mov     cl,ByteOffsetShift   ; numero di bit da spostare per
                               ;  convertire i pixel in byte

            shr     ax,cl      ; AX := offset di byte di x2
            shr     bx,cl      ; BX := offset di byte di x1
            mov     cx,ax
            sub     cx,bx      ; CX := (# byte nella linea) - 1

; rileva indirizzo di porta del controller grafico da DX

            mov     bx,dx      ; BH := maschera di bit del primo
                               ;  byte
                               ; BL := maschera di bit dell'ultimo
                               ;  byte
            mov     dx,3CEh    ; DX := porta del controller grafico
            mov     al,8       ; AL := numero reg. maschera di bit

; rende indirizzabile il buffer del video tramite DS:SI

            push    es
            pop     ds
            mov     si,di      ; DS:SI -> buffer del video

; imposta pixel nel byte all'estrema sinistra della linea
```

```
            or      bh,bh
            js      L43         ; salta se è allineato al byte (x1 è
                                ;  il pixel all'estrema sinistra nel byte)
            or      cx,cx
            jnz     L42         ; salta se c'è più di un byte nella
                                ;  linea

            and     bl,bh       ; BL := maschera di bit per la linea
            jmp     short L44

L42:        mov     ah,bh       ; AH := maschera di bit per primo
                                ;  byte
            out     dx,ax       ; aggiorna controller grafico

            movsb               ; aggiorna piani di bit
            dec     cx

; usa una veloce istruzione macchina 8086 per tracciare il resto della
; linea

L43:        mov     ah,11111111b; AH := maschera di bit
            out     dx,ax       ; aggiorna reg. di maschera di bit

            rep     movsb       ; aggiorna tutti i pixel della linea

; imposta i pixel nel byte all'estrema destra della linea

L44:        mov     ah,bl       ; AH := maschera di bit dell'ultimo
                                ;  byte
            out     dx,ax       ; aggiorna controller grafico

            movsb               ; aggiorna piani di bit

            pop     ds          ; ripristina DS
            jmp     short Lexit


; routine per dy <= dx (inclinazione <= 1)
                                ; ES:DI -> buffer del video
                                ; AL = numero reg.di maschera di bit
                                ; BL = maschera di bit per primo
                                ;  pixel
                                ; CX = # pixel da tracciare
                                ; DX = indirizzo porta controller
                                ;  grafico
                                ; SI = variabile decisionale
LoSlopeLine10:

L10:        mov     ah,bl       ; AH := maschera di bit per pixel
                                ;  successivo

L11:        or      ah,bl       ; maschera posizione corrente del
                                ;  pixel
            ror     bl,1        ; ruota valore di pixel
```

```
            jc      L14         ; salta se maschera di bit è stata
                                ;  ruotata su posizione pixel all'estrema
                                ;  sinistra

; maschera di bit non spostata fuori

            or      si,si       ; test del segno di d
            jns     L12         ; salta se> d = 0

            add     si,VARincr1; d := d + incr1
            loop    L11

            out     dx,ax       ; aggiorna reg.di maschera di bit
            or      es:[di],al  ; imposta pixel restante(i)
            jmp     short Lexit

L12:        add     si,VARincr2; d := d + incr2
            out     dx,ax       ; aggiorna reg.di maschera di bit

            or      es:[di],al  ; aggiorna piani di bit

            add     di,VARvertincr ; incrementa y
            loop    L10
            jmp     short Lexit

; maschera di bit spostata fuori

L14:        out     dx,ax       ; aggiorna reg.di maschera di bit ...

            or      es:[di],al  ; aggiorna piani di bit
            inc     di          ; incrementa x

            or      si,si       ; test del segno di d
            jns     L15         ; salta se non negativo

            add     si,VARincr1; d := d + incr1
            loop    L10
            jmp     short Lexit

L15:        add     si,VARincr2; d := d + incr2
            add     di,VARvertincr ; incremento verticale
            loop    L10
            jmp     short Lexit


; routine per dy > dx(inclinazione> 1)     ; ES:DI -> buffer del video
                                ; AH = maschera di bit per primo
                                ;  pixel
                                ; AL = numero reg.di maschera di bit
                                ; CX = # pixel da tracciare
                                ; DX = indirizzo porta controller
                                ;  grafico
                                ; SI = variabile decisionale
```

```
HiSlopeLine10:
                mov     bx,VARvertincr  ; BX := incremento y

L21:            out     dx,ax       ; aggiorna registro maschera di bit
                or      es:[di],al  ; aggiorna piani di bit

                add     di,bx       ; incrementa y

L22:            or      si,si       ; test del segno di d
                jns     L23         ; salta se d >= 0

                add     si,VARincr1 ; d := d + incr1
                loop    L21
                jmp     short Lexit


L23:            add     si,VARincr2 ; d := d + incr2

                ror     ah,1        ; ruota maschera di bit
                adc     di,0        ; incrementa DI se maschera è stata
                                    ;  ruotata su posizione di pixel
                                    ;  all'estrema sinistra

                loop    L21


; ripristina stato di default del controller grafico e ritorna al
; chiamante


Lexit:          xor     ax,ax       ; AH := 0, AL := 0
                out     dx,ax       ; ripristina reg. impostaz./azzeram.

                inc     ax          ; AH := 0, AL := 1
                out     dx,ax       ; ripristina reg. abilitazione
                                    ;  impostazione/azzeramento

                mov     al,3        ; AH := 0, AL := 3
                out     dx,ax       ; AL := # reg rotazione
                                    ;  dati/selezione funzione

                mov     ax,0FF08h   ; AH := 1111111b, AL := 8
                out     dx,ax       ; ripristina registro maschera bit

                pop     di          ; ripristina registri e ritorna
                pop     si
                mov     sp,bp
                pop     bp
                ret

Line10          ENDP

TEXT            ENDS
                END
```

Listato 67. *Una routine di tracciamento linee per le modalità grafiche EGA originali.*

All'interno dei moduli di tracciamento linee, il valore 3CEH (la porta del registro dell'indirizzo del controller grafico) viene mantenuto in DX, il valore 8 (il numero di registro di maschera di bit) viene memorizzato in AL e la maschera corrente di bit di pixel viene conservata in AH. Questo permette di aggiornare i piani di bit con solo due istruzioni in linguaggio macchina: *OUT DX,AX* per aggiornare il registro di maschera di bit e un'istruzione *MOVSB* o *OR ES:[DI],AL* che provoca un'operazione di lettura e una di scrittura di CPU.

Questa routine usa in modo accurato i registri 80x86 ed il controller grafico. L'elaborazione parallela del controller grafico aiuta la routine ad operare ad una velocità quasi pari a quella delle routine di tracciamento linee del CGA e dell'HGC.

Le modalità grafiche originali dell'EGA non utilizzano intercalazione di buffer del video, quindi la localizzazione dei pixel adiacenti verticalmente ad un dato pixel all'interno del buffer del video è un'operazione semplice. Se ogni linea contiene *n* byte di pixel, il pixel immediatamente sovrastante ad un dato pixel è distante *-n* byte, ed il pixel immediatamente sottostante è distante *n* byte. Il codice per incrementare gli indirizzi di pixel verticalmente è quindi più semplice del codice corrispondente per CGA o HGC (confrontate, ad esempio, il codice del loop alla label *L32* dei Listati 64 e 67).

Il controller grafico gestisce autonomamente una qualsiasi delle quattro operazioni di pixel (*XOR, AND, OR* e sostituzione), quindi l'unico codice supplementare richiesto per supportare queste funzioni è costituito da alcune istruzioni per caricare il registro di rotazione dati/selezione funzione (03H). Questo compito fa parte del codice di configurazione controller grafico nella prima parte della routine del Listato 67.

CONSIGLIO

Potete utilizzare questa routine di tracciamento linee nelle modalità a 4 colori 640 per 350 e in quelle monocromatiche. Assicuratevi di specificare il corretto valore di pixel in queste modalità in modo che la routine imposti i bit nei corretti piani di bit (vedere Capitolo 4).

MCGA

Su MCGA, nelle modalità compatibili CGA, potete utilizzare le routine di tracciamento linee del CGA. Le modalità non CGA (la modalità a 2 colori 640 per 480 e quella a 256 colori 320 per 200) richiedono proprie routine, come mostrato nei Listati 68 e 69, ma queste sono facilmente ricavabili dal codice per la modalità a 2 colori 640 per 200.

```
                TITLE   'Listato 68'
                NAME    Line11
                PAGE    55,132

;
; Nome:         Line11
;
; Funzione:     Traccia una linea nella modalità a 2 colori 640x480
;                (MCGA, VGA)
; Chiamante:    Microsoft C:
;
;                       azzera Line11(x1,y1,x2,y2,n);
;
;                       int x1,y1,x2,y2;            /* coordinate del pixel */
;
;                       int n;                      /* valore di pixel */
;

ARGx1           EQU     word ptr [bp+4] ; indirizzamento cornice di stack
ARGy1           EQU     word ptr [bp+6]
ARGx2           EQU     word ptr [bp+8]
ARGy2           EQU     word ptr [bp+10]
ARGn            EQU     byte ptr [bp+12]
VARincr1        EQU     word ptr [bp-2]
VARincr2        EQU     word ptr [bp-4]
VARroutine      EQU     word ptr [bp-6]

BytesPerLine    EQU     80          ; byte in una riga di pixel
ByteOffsetShift EQU         3       ; usato per convertire i pixel in
                                    ;  offset di byte

DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr10:near

                PUBLIC  _Line11
_Line11         PROC    near

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                sub     sp,6        ; spazio di stack per le variabili
                                    ;  locali
                push    si
                push    di

; verifica se si tratta di linea verticale

                mov     si,BytesPerLine ; SI := incremento y iniziale

                mov     cx,ARGx2
                sub     cx,ARGx1    ; CX := x2 - x1
                jz      VertLine11  ; salta se linea verticale

; forza x1 < x2
```

```
            jns     L01         ; salta se x2>0 x1

            neg     cx          ; CX := x1 - x2

            mov     bx,ARGx2    ; scambia x1 e x2
            xchg    bx,ARGx1
            mov     ARGx2,bx

            mov     bx,ARGy2    ; scambia y1 e y2
            xchg    bx,ARGy1
            mov     ARGy2,bx

; calcola dy = ABS(y2-y1)

L01:        mov     bx,ARGy2
            sub     bx,ARGy1    ; BX := y2 - y1
            jnz     L02

            jmp     HorizLine11; salta se linea orizzontale

L02:        jns     L03

            neg     bx          ; BX := y1 - y2
            neg     si          ; nega incremento y

; seleziona routine appropriata per inclinazione linea

L03:        mov     VARroutine,offset LoSlopeLine11
            cmp     bx,cx
            jle     L04         ; salta se dy <= dx (inclinaz.<= 1)
            mov     VARroutine,offset HiSlopeLine11
            xchg    bx,cx       ; scambia dy e dx

; calcola variabile decisionale iniziale e incrementa

L04:        shl     bx,1        ; BX := 2 * dy
            mov     VARincr1,bx; incr1 := 2 * dy
            sub     bx,cx
            mov     di,bx       ; DI := d = 2 * dy - dx
            sub     bx,cx
            mov     VARincr2,bx; incr2 := 2 * (dy - dx)

; calcola primo indirizzo di pixel

            push    cx          ; mantiene questo registro
            mov     ax,ARGy1    ; AX := y
            mov     bx,ARGx1    ; BX := x
            call    PixelAddr10; AH := maschera di bit
                                ; ES:BX -> buffer
                                ; CL := # bit da spostare a sinistra

            mov     al,ARGn     ; AL := valore di pixel non spostato
            shl     ax,cl       ; AH := maschera di bit nella
                                ;  posizione corretta
                                ; AL := valore di pixel nella
                                ;  posizione corretta

            mov     dx,ax       ; DH := maschera di bit
```

```
                                ; DL := valore di pixel
            not     dh          ; DH := inverte maschera di bit

            pop     cx          ; ripristina questo registro
            inc     cx          ; CX := # di pixel da tracciare

L05:        jmp     VARroutine  ; salta a routine appropriata per
                                ;  inclinazione

; routine per linee verticali

VertLine11: mov     ax,ARGy1    ; AX := y1
            mov     bx,ARGy2    ; BX := y2
            mov     cx,bx
            sub     cx,ax       ; CX := dy
            jge     L31         ; salta se dy >= 0

            neg     cx          ; forza dy> = 0
            mov     ax,bx       ; AX := y2

L31:        inc     cx          ; CX := # di pixel da tracciare
            mov     bx,ARGx1    ; BX := x
            push    cx          ; mantiene questo registro
            call    PixelAddr10; AH := maschera di bit
                                ; ES:BX -> buffer del video
                                ; CL := # bit da spostare a sinistra
            mov     al,ARGn     ; AL := valore di pixel
            shl     ax,cl       ; AH := maschera di bit nella
                                ;  posizione corretta
                                ; AL := valore di pixel nella
                                ;  posizione corretta
            not     ah          ; AH := inverte maschera di bit
            pop     cx          ; ripristina questo registro

; traccia la linea

            test    al,al
            jz      L33         ; salta se valore di pixel = 0

L32:        or      es:[bx],al  ; imposta valori di pixel nel buffer
            add     bx,si
            loop    L32
            jmp     short L34

L33:        and     es:[bx],ah  ; azzera valori di pixel nel buffer
            add     bx,si
            loop    L33

L34:        jmp     Lexit


; routine per linee orizzontali (inclinazione = 0)

HorizLine11: mov    ax,ARGy1
            mov     bx,ARGx1
            call    PixelAddr10; AH := maschera di bit
                                ; ES:BX -> buffer del video
                                ; CL := # bit da spostare a sinistra
```

```
            mov     di,bx       ; ES:DI -> buffer

            mov     dh,ah
            not     dh          ; DH := maschera di bit non spostata
                                ;       per byte all'estrema sinistra
            mov     dl,0FFh     ; DL := maschera di bit non spostata
                                ;       per byte all'estrema destra

            shl     dh,cl       ; DH := inverte maschera di bit per
                                ;  primo byte
            not     dh          ; DH := maschera di bit per primo
                                ;  byte

            mov     cx,ARGx2
            and     cl,7
            xor     cl,7        ; CL := numero di bit da spostare a
                                ;  sinistra
            shl     dl,cl       ; DL := maschera di bit per ultimo
                                ;  byte

; determina offset byte del primo e dell'ultimo pixel della linea

            mov     ax,ARGx2    ; AX := x2
            mov     bx,ARGx1    ; BX := x1

            mov     cl,ByteOffsetShift    ; numero di bit da spostare per
                                          ;  convertire i pixel in byte

            shr     ax,cl       ; AX := offset di byte di x2
            shr     bx,cl       ; BX := offset di byte di x1
            mov     cx,ax
            sub     cx,bx       ; CX := (# byte nella linea) - 1

; estende valore di pixel per tutto un byte

            mov     bx,offset DGROUP:PropagatedPixel
            mov     al,ARGn     ; AL := valore di pixel
            xlat

; imposta pixel nel byte all'estrema sinistra della linea

            or      dh,dh
            js      L43         ; salta se è allineato al byte (x1 è
                                ;  il pixel all'estrema sinistra nel
                                ;  byte)
            or      cx,cx
            jnz     L42         ; salta se c'è più di un byte nella
                                ;  linea

            and     dl,dh       ; maschera di bit per la linea
            jmp     short L44

L42:        mov     ah,al
            and     ah,dh       ; AH := bit mascherati di pixel
            not     dh          ; DH := inverte maschera di bit per
                                ;  primo byte
            and     es:[di],dh  ; azzera pixel mascherati nel buffer
```

```
            or      es:[di],ah ; aggiorna pixel mascherati nel
                               ;  buffer
            inc     di
            dec     cx

; usa una veloce istruzione macchina 8086 per tracciare il resto della
; linea

L43:        rep     stosb      ; aggiorna tutti i pixel nella linea

; imposta pixel nel byte all'estrema destra della linea

L44:        and     al,dl      ; AL := pixel mascherati per ultimo
                               ;  byte
            not     dl
            and     es:[di],dl ; azzera pixel mascherati nel buffer
            or      es:[di],al ; aggiorna pixel mascherati nel
                               ;  buffer

            jmp     Lexit


; routine per dy <= dx (inclinazione <= 1) ; ES:BX -> buffer del video
                               ; CX = # pixel da tracciare
                               ; DH = inverte maschera di bit
                               ; DL = valore di pixel nella
                               ;  posizione corretta
                               ; SI = byte per riga di pixel
                               ; DI = variabile decisionale
LoSlopeLine11:

L10:        mov     ah,es:[bx] ; AH := byte del buffer del video

L11:        and     ah,dh      ; azzera valore di pixel all'offset
                               ;  corrente di bit
            or      ah,dl      ; imposta valore di pixel nel byte

            ror     dl,1       ; ruota valore di pixel
            ror     dh,1       ; ruota maschera di bit
            jnc     L14        ; salta se maschera di bit è stata
                               ;  ruotata sulla posizione di pixel
                               ;  all'estrema sinistra

; maschera di bit non spostata fuori

            or      di,di      ; test del segno di d
            jns     L12        ; salta se d> = 0

            add     di,VARincr1; d := d + incr1
            loop    L11

            mov     es:[bx],ah ; memorizza restanti pixel nel buffer
            jmp     short Lexit

L12:        add     di,VARincr2; d := d + incr2
            mov     es:[bx],ah ; aggiorna buffer

            add     bx,si      ; incrementa y
```

```
                loop    L10
                jmp     short Lexit

; maschera di bit spostata fuori

L14:            mov     es:[bx],ah ; aggiorna buffer
                inc     bx         ; BX := offset del byte successivo

                or      di,di      ; test del segno di d
                jns     L15        ; salta se non negativo

                add     di,VARincr1; d := d + incr1
                loop    L10
                jmp     short Lexit

L15:            add     di,VARincr2; d := d + incr2

                add     bx,si      ; incrementa y
                loop    L10
                jmp     short Lexit

; routine per dy>  dx (inclinazione> 1)    ; ES:BX -> buffer del video
                                   ; CX = # pixel da tracciare
                                   ; DH = inverte maschera di bit
                                   ; DL = valore di pixel nella
                                   ;  posizione corretta
                                   ; SI = byte per riga di pixel
                                   ; DI = variabile decisionale
HiSlopeLine11:

L21:            and     es:[bx],dh ; azzera valore di pixel nel buffer
                                   ;  del video
                or      es:[bx],dl ; imposta valore di pixel nel byte

                add     bx,si      ; incrementa y

L22:            or      di,di      ; test del segno di d
                jns     L23        ; salta se d> = 0

                add     di,VARincr1; d := d + incr1
                loop    L21

                jmp     short Lexit


L23:            add     di,VARincr2; d := d + incr2

                ror     dl,1       ; ruota valore di pixel
                ror     dh,1       ; ruota maschera di bit
                cmc                ; cf impostato se maschera di bit non
                                   ;  è stata ruotata su posizione di
                                   ;  pixel all'estrema sinistra

                adc     bx,0       ; BX := offset del byte successivo

                loop    L21


Lexit:          pop     di
```

```
                pop     si
                mov     sp,bp
                pop     bp
                ret

_Line11         ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

PropagatedPixel DB    00000000b  ; 0
                DB    11111111b  ; 1

_DATA             ENDS

                  END
```

Listato 68. *Una routine di tracciamento linee per la modalità a 2 colori 640 per 480 dell'MCGA e della VGA.*

```
                TITLE   'Listato 69'
                NAME    Line13
                PAGE    55,132

;
; Nome:         Line13
;
; Funzione:     Traccia una linea nella modalità a 256 colori 320x200 MCGA/VGA
;
; Chiamante:    Microsoft C:
;
;                       azzera Line13(x1,y1,x2,y2,n);
;
;                       int x1,y1,x2,y2;            /* coordinate del pixel */
;
;                       int n;                      /* valore di pixel */
;

ARGx1           EQU     word ptr [bp+4] ; indirizzamento cornice di stack
ARGy1           EQU     word ptr [bp+6]
ARGx2           EQU     word ptr [bp+8]
ARGy2           EQU     word ptr [bp+10]
ARGn            EQU     byte ptr [bp+12]
VARincr1        EQU     word ptr [bp-2]
VARincr2        EQU     word ptr [bp-4]
VARroutine      EQU     word ptr [bp-6]

BytesPerLine    EQU     320

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr13:near
```

```
                PUBLIC _Line13
_Line13         PROC   near

                push   bp          ; mantiene registri del chiamante
                mov    bp,sp
                sub    sp,6        ; spazio di stack per le variabili
                                   ;  locali
                push   si
                push   di

; verifica se si tratta di linea verticale

                mov    si,BytesPerLine; incremento y iniziale

                mov    cx,ARGx2
                sub    cx,ARGx1    ; CX := x2 - x1
                jz     VertLine13 ; salta se linea verticale

; forza x1<  x2

                jns    L01         ; salta se x2 > x1

                neg    cx          ; CX := x1 - x2

                mov    bx,ARGx2    ; scambia x1 e x2
                xchg   bx,ARGx1
                mov    ARGx2,bx

                mov    bx,ARGy2    ; scambia y1 e y2
                xchg   bx,ARGy1
                mov    ARGy2,bx

; calcola dy = ABS(y2-y1)

L01:            mov    bx,ARGy2
                sub    bx,ARGy1    ; BX := y2 - y1
                jz     HorizLine13; salta se linea orizzontale

                jns    L03         ; salta se inclinazione è positiva

                neg    bx          ; BX := y1 - y2
                neg    si          ; nega incremento y

; seleziona routine appropriata per inclinazione linea

L03:            push   si          ; mantiene incremento y

                mov    VARroutine,offset LoSlopeLine13
                cmp    bx,cx
                jle    L04         ; salta se dy <= dx (inclinaz.<= 1)
                mov    VARroutine,offset HiSlopeLine13
                xchg   bx,cx       ; scambia dy e dx

; calcola variabile decisionale iniziale e incrementa

L04:            shl    bx,1        ; BX := 2 * dy
                mov    VARincr1,bx; incr1 := 2 * dy
```

```
            sub     bx,cx
            mov     si,bx       ; SI := d = 2 * dy - dx
            sub     bx,cx
            mov     VARincr2,bx; incr2 := 2 * (dy - dx)
; calcola primo indirizzo di pixel

            push    cx          ; mantiene questo registro
            mov     ax,ARGy1    ; AX := y
            mov     bx,ARGx1    ; BX := x
            call    PixelAddr13; ES:BX -> buffer

            mov     di,bx       ; ES:DI -> buffer

            pop     cx          ; ripristina questo registro
            inc     cx          ; CX := # di pixel da tracciare

            pop     bx          ; BX := incremento y
            jmp     VARroutine  ; salta a routine appropriata per
                                ;  inclinazione


; routine per linee verticali

VertLine13:  mov    ax,ARGy1    ; AX := y1
            mov     bx,ARGy2    ; BX := y2
            mov     cx,bx
            sub     cx,ax       ; CX := dy
            jge     L31         ; salta se dy >= 0

            neg     cx          ; forza dy> = 0
            mov     ax,bx       ; AX := y2

L31:        inc     cx          ; CX := # di pixel da tracciare
            mov     bx,ARGx1    ; BX := x
            push    cx          ; mantiene questo registro
            call    PixelAddr13; ES:BX -> buffer del video
            pop     cx

            mov     di,bx       ; ES:DI -> buffer del video
            dec     si          ; SI := byte/linea -1

            mov     al,ARGn     ; AL := valore di pixel

L32:        stosb               ; imposta valore di pixel nel buffer
            add     di,si       ; incrementa a riga successiva
            loop    L32

            jmp     short Lexit


; routine per linee orizzontali (inclinazione = 0)

HorizLine13:
            push    cx          ; mantiene CX
            mov     ax,ARGy1
            mov     bx,ARGx1
            call    PixelAddr13; ES:BX -> buffer del video
```

```
            mov     di,bx       ; ES:DI -> buffer

            pop     cx
            inc     cx          ; CX := numero di pixel da tracciare

            mov     al,ARGn     ; AL := valore di pixel

            rep     stosb       ; aggiorna il buffer del video

            jmp     short Lexit
; routine per dy <= dx (inclinazione <= 1) ; ES:DI -> buffer del video
                                ; BX = incremento y
                                ; CX = # pixel da tracciare
                                ; SI = variabile decisionale
LoSlopeLine13:

            mov AL,ARGn         ; AL := valore del pixel

L11:        stosb               ; memorizza pixel, incrementa x

            or      si,si       ; test del segno di d
            jns     L12         ; salta se d> = 0

            add     si,VARincr1 ; d := d + incr1
            loop    L11
            jmp     short Lexit

L12:        add     si,VARincr2 ; d := d + incr2
            add     di,bx       ; incrementa y
            loop    L11
            jmp     short Lexit


; routine per dy> dx (inclinazione> 1)     ; ES:DI -> buffer del video
                                ; BX = incremento y
                                ; CX = # pixel da tracciare
                                ; SI = variabile decisionale
HiSlopeLine13:

            mov     al,ARGn     ; AL := valore di pixel

L21:        stosb               ; aggiorna pixel successivo,
                                ;  incrementa x

            add     di,bx       ; incrementa y

L22:        or      si,si       ; test del segno di d
            jns     L23         ; salta se d> = 0

            add     si,VARincr1 ; d := d + incr1
            dec     di          ; decrementa x (già incrementato da
                                ;  stosb)
            loop    L21
            jmp     short Lexit
```

```
L23:            add     si,VARincr2; d := d + incr2
                loop    L21


Lexit:          pop     di          ; ripristina registri e ritorna
                pop     si
                mov     sp,bp
                pop     bp
                ret

_Line13         ENDP

_TEXT           ENDS

                END
```

Listato 69. *Una routine di tracciamento linee per la modalità a 256 colori 320 per 200 dell'MCGA e della VGA.*

VGA

Una volta implementate le routine per EGA e per MCGA, potete tracciare linee in qualsiasi modalità grafica VGA. Per tracciare linee nella modalità a 16 colori 640 per 480, usate la routine per 16 colori 640 per 350.

Scheda InColor

Dal momento che l'indirizzamento dei pixel nel buffer del video è identico sia sulla scheda InColor sia sulle schede monocromatiche Hercules, l'unica differenza significativa nelle routine di tracciamento linee per la scheda InColor, come vedrete nel Listato 70, è costituita dal codice supplementare per selezionare il valore di pixel specificato. Si noti come viene utilizzata la modalità di scrittura 1 della scheda InColor insieme ad un appropriato valore di primo piano nel registro di colore lettura/scrittura per impostare i valori dei pixel adiacenti in ogni byte del buffer. Questa tecnica è simile all'uso della modalità di scrittura 0 e del registro di impostazione/azzeramento su EGA.

```
                TITLE   'Listato 70'
                NAME    LineInC
                PAGE    55,132

;
; Nome:         LineInC
;
; Funzione:     Traccia una linea nella modalità a 16 colori 720x348 InColor
; Hercules
;
; Chiamante:    Microsoft C:
```

```
;
;                    azzera LineInC(x1,y1,x2,y2,n);
;
;                    int x1,y1,x2,y2;           /* coordinate del pixel */
;
;                    int n;                     /* valore di pixel */
;

ARGx1          EQU    word ptr [bp+4] ; indirizzamento cornice di stack
ARGy1          EQU    word ptr [bp+6]
ARGx2          EQU    word ptr [bp+8]
ARGy2          EQU    word ptr [bp+10]
ARGn           EQU    byte ptr [bp+12]
VARleafincr    EQU    word ptr [bp-2]
VARincr1       EQU    word ptr [bp-4]
VARincr2       EQU    word ptr [bp-6]
VARroutine     EQU    word ptr [bp-8]

ByteOffsetShift EQU  3          ; usato per convertire i pixel in
                                ;  offset di byte
DefaultRWColorEQU    0Fh        ; valore di default registro colore
                                ;  lettura/scrittura

_TEXT          SEGMENTbyte public 'CODE'
               ASSUME cs:_TEXT

               EXTRN  PixelAddrHGC:near

               PUBLIC _LineInC
_LineInC       PROC   near

               push   bp         ; mantiene registri del chiamante
               mov    bp,sp
               sub    sp,8       ; spazio di stack per le variabili
                                 ;  locali
               push   si
               push   di

               mov    si,2000h   ; incremento per intercalazione di
                                 ;  buffer del video
               mov    di,90-8000h; incremento dall'ultima alla prima
                                 ;  intercalazione

; imposta registri del CRTC di InColor

               mov    dx,3B4h    ; DX := porta di I/O del CRTC
               mov    ax,5F19h   ; AH bit 6 := 1 (polarità di
                                 ;  maschera)
                                 ; AH bit 5-4 := 1 (modalità di
                                 ;  scrittura)
                                 ; AH bit 3-0 := bit "ininfluenti"
                                 ; AL := numero registro di controllo
                                 ;  lettura/scrittura
               out    dx,ax      ; imposta registro di controllo
                                 ;  lettura/scrittura

               inc    ax         ; AL := 1Ah (numero registro colore
                                 ;  lettura/scrittura)
```

```
            mov     ah,ARGn     ; AH := valore primo piano
            out     dx,ax       ; imposta registro colore
                                ;  lettura/scrittura

            mov     cx,ARGx2
            sub     cx,ARGx1    ; CX := x2 - x1
            jz      VertLineInC; salta se linea verticale

; forza x1 < x2

            jns     L01         ; salta se x2 > x1

            neg     cx          ; CX := x1 - x2

            mov     bx,ARGx2    ; scambia x1 e x2
            xchg    bx,ARGx1
            mov     ARGx2,bx

            mov     bx,ARGy2    ; scambia y1 e y2
            xchg    bx,ARGy1
            mov     ARGy2,bx

; calcola dy = ABS(y2-y1)

L01:        mov     bx,ARGy2
            sub     bx,ARGy1    ; BX := y2 - y1
            jz      HorizLineInC    ; salta se linea orizzontale

            jns     L03

            neg     bx          ; BX := y1 - y2
            neg     si          ; nega incrementi per intercalazione
                                ;  buffer
            neg     di

; seleziona routine appropriata per inclinazione linea

L03:        mov     VARleafincr,di  ; salva incremento per intercalazione
                                    ;  buffer

            mov     VARroutine,offset LoSlopeLineInC
            cmp     bx,cx
            jle     L04         ; salta se dy <= dx (inclinaz.<= 1)
            mov     VARroutine,offset HiSlopeLineInC
            xchg    bx,cx       ; scambia dy e dx

; calcola variabile decisionale iniziale e incrementa

L04:        shl     bx,1        ; BX := 2 * dy
            mov     VARincr1,bx; incr1 := 2 * dy
            sub     bx,cx
            mov     di,bx       ; DI := d = 2 * dy - dx
            sub     bx,cx
            mov     VARincr2,bx; incr2 := 2 * (dy - dx)

; calcola primo indirizzo di pixel
```

```
            push    cx          ; mantiene questo registro
            mov     ax,ARGy1    ; AX := y
            mov     bx,ARGx1    ; BX := x
            call    PixelAddrHGC    ; AH := maschera di bit
                                ; ES:BX -> buffer
                                ; CL := # bit da spostare a sinistra

            shl     ah,cl
            mov     dh,ah       ; DH := maschera di bit nella
                                ;  posizione corretta

            pop     cx          ; ripristina questo registro
            inc     cx          ; CX := # di pixel da tracciare

            jmp     VARroutine  ; salta a routine appropriata per
                                ;  inclinazione

; routine per linee verticali

VertLineInC:mov     ax,ARGy1    ; AX := y1
            mov     bx,ARGy2    ; BX := y2
            mov     cx,bx
            sub     cx,ax       ; CX := dy
            jge     L31         ; salta se dy >= 0

            neg     cx          ; forza dy> = 0
            mov     ax,bx       ; AX := y2

L31:        inc     cx          ; CX := # di pixel da tracciare
            mov     bx,ARGx1    ; BX := x
            push    cx          ; mantiene questo registro
            call    PixelAddrHGC    ; AH := maschera di bit
                                ; ES:BX -> buffer del video
                                ; CL := # bit da spostare a sinistra
            shl     ah,cl       ; AH := maschera di bit nella
                                ;  posizione corretta
            pop     cx          ; ripristina questo registro

L32:        or      es:[bx],ah  ; imposta valori di pixel nel buffer

            add     bx,si       ; incrementa all'area successiva di
                                ;  intercalazione
            jns     L33

            add     bx,di       ; incrementa alla prima area
                                ;  d'intercalazione
L33:        loop    L32

            jmp     Lexit


; routine per linee orizzontali (inclinazione = 0)

HorizLineInC:mov    ax,ARGy1
            mov     bx,ARGx1
            call    PixelAddrHGC    ; AH := maschera di bit
                                ; ES:BX -> buffer del video
```

```
                                ; CL := # bit da spostare a sinistra
            mov     di,bx       ; ES:DI -> buffer

            mov     dh,ah
            not     dh          ; DH := maschera di bit non spostata
                                ;       per byte all'estrema sinistra
            mov     dl,0FFh     ; DL := maschera di bit non spostata
                                ;       per byte all'estrema destra

            shl     dh,cl       ; DH := inverte maschera di bit per
                                ;  primo byte
            not     dh          ; DH := maschera di bit per primo
                                ;  byte

            mov     cx,ARGx2
            and     cl,7
            xor     cl,7        ; CL := numero di bit da spostare a
                                ;  sinistra
            shl     dl,cl       ; DL := maschera di bit per ultimo
                                ;  byte

; determina offset byte del primo e dell'ultimo pixel della linea

            mov     ax,ARGx2    ; AX := x2
            mov     bx,ARGx1    ; BX := x1

            mov     cl,ByteOffsetShift    ; numero di bit da spostare per
                                          ;  convertire i pixel in byte

            shr     ax,cl       ; AX := offset di byte di x2
            shr     bx,cl       ; BX := offset di byte di x1
            mov     cx,ax
            sub     cx,bx       ; CX := (# byte nella linea) - 1

; imposta pixel nel byte all'estrema sinistra della linea

            or      dh,dh
            js      L43         ; salta se è allineato al byte (x1 è
                                ;  il pixel all'estrema sinistra nel
                                ;  byte)
            or      cx,cx
            jnz     L42         ; salta se c'è più di un byte nella
                                ;  linea

            and     dl,dh       ; maschera di bit per la linea
            jmp     short L44

L42:        or      es:[di],dh  ; aggiorna pixel mascherati nel
                                ;  buffer
            inc     di
            dec     cx

; usa una veloce istruzione macchina 8086 per tracciare il resto della
; linea

L43:        mov     al,0FFh     ; maschera di bit a 8 pixel
            rep     stosb       ; aggiorna tutti i pixel nella linea
```

```
; imposta pixel nel byte all'estrema destra della linea

L44:            or      es:[di],dl ; aggiorna pixel mascherati nel
                                   ;  buffer
                jmp     Lexit


; routine per dy <= dx (inclinazione <= 1) ; ES:BX -> buffer del video
                                   ; CX = # pixel da tracciare
                                   ; DH = maschera di bit
                                   ; SI = incremento dell'intercalazione
                                   ;  del buffer
                                   ; DI = variabile decisionale
LoSlopeLineInC:

L10:            mov     ah,es:[bx] ; piani di bit di latch
                                   ; AH := 0 perché tutti i piani sono
                                   ;  "ininfluenti"

L11:            or      ah,dh      ; imposta valore di pixel nel byte

                ror     dh,1       ; ruota maschera di bit
                jc      L14        ; salta se maschera di bit è stata
                                   ;  ruotata sulla posizione di pixel
                                   ;  all'estrema sinistra

; maschera di bit non spostata fuori

                or      di,di      ; test del segno di d
                jns     L12        ; salta se d >= 0

                add     di,VARincr1; d := d + incr1
                loop    L11

                mov     es:[bx],ah ; memorizza restanti pixel nel
                                   ;  buffer
                jmp     short Lexit

L12:            add     di,VARincr2; d := d + incr2
                mov     es:[bx],ah ; aggiorna buffer

                add     bx,si      ; incrementa y
                jns     L13        ; salta se non si trova nell'ultima
                                   ;  intercalazione

                add     bx,VARleafincr ; incrementa nell'intercalazione
                                       ;  successiva

L13:            loop    L10
                jmp     short Lexit

; maschera di bit spostata fuori

L14:            mov     es:[bx],ah ; aggiorna buffer
                inc     bx         ; BX := offset del byte successivo

                or      di,di      ; test del segno di d
                jns     L15        ; salta se non negativo
```

```
                add     di,VARincr1; d := d + incr1
                loop    L10
                jmp     short Lexit

L15:            add     di,VARincr2; d := d + incr2

                add     bx,si       ; incrementa y
                jns     L16         ; salta se non si trova nell'ultima
                                    ;  intercalazione

                add     bx,VARleafincr ; incrementa nell'intercalazione
                                    ;  successiva

L16:            loop    L10         ; loop fino a che tutti i pixel sono
                                    ;  impostati
                jmp     short Lexit

; routine per dy > dx (inclinazione> 1)    ; ES:BX -> buffer del video
                                    ; CX = # pixel da tracciare
                                    ; DH = maschera di bit
                                    ; SI = incremento dell'intercalazione
                                    ;  del buffer
                                    ; DI = variabile decisionale
HiSlopeLineInC:

L21:            or      es:[bx],dh ; imposta valore di pixel nel buffer
                                    ;  del video

                add     bx,si       ; incrementa y
                jns     L22         ; salta se non si trova nell'ultima
                                    ;  intercalazione

                add     bx,VARleafincr ; incrementa nell'intercalazione
                                        ;  successiva

L22:            or      di,di
                jns     L23         ; salta se d = 0

                add     di,VARincr1; d := d + incr1
                loop    L21
                jmp     short Lexit


L23:            add     di,VARincr2; d := d + incr2

                ror     dh,1        ; ruota maschera di bit
                adc     bx,0        ; BX := offset del byte successivo
                                    ;  (incrementato se maschera di bit è
                                    ;  stata ruotata sulla posizione di
                                    ;  pixel all'estrema sinistra)

                loop    L21

Lexit:          mov     dx,3B4h     ; DX := porta di I/O del CRTC
                mov     ax,0F18h
                out     dx,ax       ; ripristina valore di default di
                                    ;  maschera di piano
```

```
                mov     ax,4019h    ; ripristina valore di default di
                                    ;  controllo lettura/scrittura
                out     dx,ax

                inc     ax          ; ripristina valore di default di
                                    ;  colore lettura/scrittura
                mov     ah,DefaultRWColor
                out     dx,ax

                pop     di          ; ripristina registri e ritorna
                pop     si
                mov     sp,bp
                pop     bp
                ret

_LineInC        ENDP

_TEXT           ENDS

                END
```

Listato 70. *Una routine di tracciamento linee per la modalità a 16 colori 720 per 348 della scheda InColor.*

Attributi di linea

L'algoritmo di tracciamento linee visto in questo capitolo traccia linee che hanno una larghezza corrispondente esattamente ad un pixel. Di conseguenza, le linee diagonali appaiono meno evidenziate delle linee orizzontali e verticali. Potete aumentare lo spessore delle linee diagonali modificando il loop interno di impostazione pixel di una routine di tracciamento linee di Bresenham in modo che imposti sempre il pixel *B* prima di selezionare il pixel successivo della linea. Le linee risultanti saranno più spesse, ma la routine modificata opererà ad una velocità più ridotta in quanto dovrà aggiornare più pixel, in particolare nelle linee con inclinazioni vicine a 1 o -1.

Per tracciare linee più spesse, tracciate semplicemente linee parallele adiacenti e contigue. Se state utilizzando un dispositivo di puntamento per tracciare in modo interattivo una linea spessa, utilizzate una serie di linee orizzontali o verticali adiacenti. Dopo aver implementato una veloce routine per il tracciamento di linee orizzontali, potete scrivere una routine ad alto livello che tracci linee spesse richiamando la routine di linee orizzontali originale.

In alcune applicazioni, potreste desiderare di tracciare delle linee tratteggiate o delle linee multicolore che contengano dei modelli di valori di pixel. Per fare ciò, modificate il loop interno della vostra routine di tracciamento linee in modo che selezioni i valori di pixel da una lista circolare di valori possibili. Ruotate la lista ogni volta che impostate un pixel.

Definizione della linea

Nessuna delle routine in linguaggio Assembler di questo capitolo conferma le coordinate di pixel da voi fornite come estremità delle linee. Ad esempio, se richiamate la routine per la modalità a 2 colori 640 per 200 per tracciare una linea da (0,0) a (1000,1000), la routine aggiorna senza esitazione circa 800 pixel in locazioni di memoria che non esistono nella RAM video disponibile, per tutta la distanza da (200,200) a (1000,1000). Per evitare questo tipo di errore, dovete determinare quale parte di una qualsiasi linea arbitraria si trovi all'interno di una data area del buffer del video. Questo processo potrebbe essere indicato come definizione della linea.

Nel caso della modalità a 2 colori 640 per 200, l'area nella quale le linee devono essere definite corrisponde al rettangolo definito da (0,0) nell'angolo superiore sinistro e da (639,199) nell'angolo inferiore destro. Definirete quindi una linea con estremità a (0,0) e (1000,1000) in modo che venga tracciata solo l'area da (0,0) a (199,199). Evitando l'errore di aggiornare della RAM inesistente, potreste anche migliorare le prestazioni del vostro programma, dal momento che la routine originale di tracciamento linee non tenterà di aggiornare quei pixel inesistenti.

Definizione della linea pixel per pixel

Un approccio semplicistisco alla definizione della linea è rappresentato dall'inserimento di un test di definizione linea nel loop interno delle vostre routine di tracciamento linee. Subito prima di impostare il valore di ogni pixel, la vostra routine potrebbe confrontare la maschera di bit di pixel corrente e l'indirizzo di buffer con una serie di limiti precalcolati. Se l'indirizzo, la maschera di bit o entrambi superano i limiti, la routine non aggiornerà il buffer del video. Tuttavia, le operazioni di servizio necessarie per la definizione della linea usando questa procedura potrebbero essere più lunghe del tempo necessario per calcolare e tracciare i pixel della linea.

CONSIGLIO

In generale, evitate di incorporare il codice per la definizione della linea nelle routine di tracciamento linee a basso livello, a prescindere da quanto possa essere efficiente il codice. Il tenere le funzioni separate può migliorare le prestazioni, dal momento che un'applicazione può richiamare direttamente le routine di tracciamento linee, aggirando completamente il codice di definizione linee quando non è necessario.

Un approccio violento

Un altro metodo per definire una linea è rappresentato dall'uso dell'equazione relativa per calcolare dove, eventualmente, il segmento di linea da tracciare si intersechi con i bordi dell'area di visualizzazione rettangolare. Ad esempio, nella Figura 73 l'inclinazione *m* della linea è

```
m = dy/dx = (y2-y1)/(x2-x1)n= (100-40)/(750-150) = 0.1
```

Il punto di intersezione *y* può essere calcolato sostituendo *x1* e *y1* nell'equazione della linea:

```
b = y1-m*x1 = 40-(0.1*150) = 25
```

L'equazione della linea è quindi

```
y = 0.1*x+25
```

Per calcolare le coordinate dell'intersezione tra la linea ed i bordi della finestra, sostituite le coordinate *x* dei bordi verticali della finestra e le coordinate *y* dei suoi bordi orizzontali nell'equazione. Ogni volta che risolvete l'equazione per un lato del rettangolo, controllate il risultato per verificare se il punto di intersezione si trovi effettivamente all'interno del segmento di linea da tracciare oltre che all'interno del rettangolo.

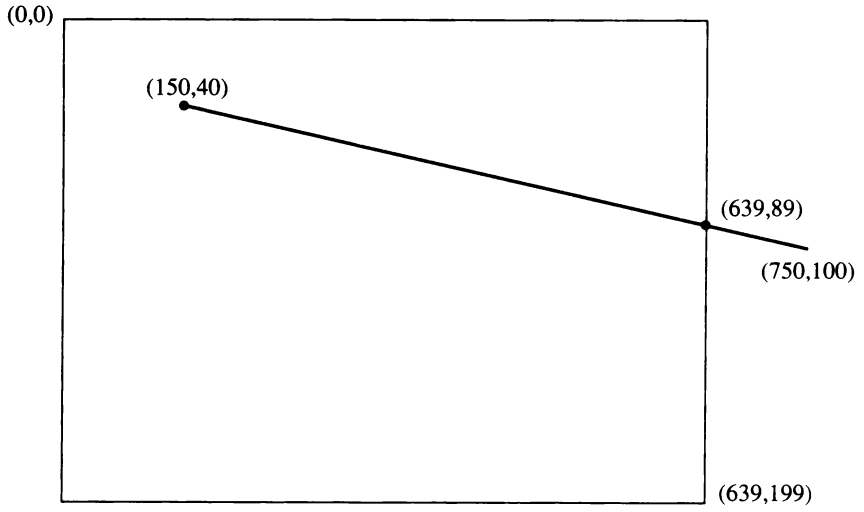


Figura 73. *Segmento di linea (150,40)-(750,100) definito a (639,89) in modalità a 2 colori 640 per 200.*

Questo approccio alla definizione di linea comporta molti calcoli, principalmente perché l'equazione della linea deve essere risolta quattro volte per ogni segmento di linea definito. Dovete inoltre confrontare i risultati con i limiti del segmento di linea per determinare se i punti di intersezione si trovino tra le estremità. Inoltre, dovete anche gestire casi particolari come ad esempio linee orizzontali e verticali o "linee" anormali costituite da un singolo pixel. Questo carico di calcoli rende poco pratico l'approccio violento alla definizione di linea.

Un migliore algoritmo

Un algoritmo più efficiente per la definizione di linea confronta le estremità del segmento di linea con i confini dell'area rettangolare prima di calcolare i punti di intersezione. Di conseguenza, vengono eseguiti pochi calcoli per le linee che non richiedono alcuna definizione. L'algoritmo di Sutherland-Cohen, che utilizza questo approccio, è ampiamente conosciuto grazie alla sua semplicità e alla sua efficienza nei calcoli.

Concettualmente, l'algoritmo estende i bordi dell'area rettangolare di definizione dividendo il piano in nove aree (vedere Figura 74). Ogni estremità del segmento di linea da definire ricade in una di queste aree. L'identificazione dell'area che corrisponde ad ogni estremità rende facile la determinazione della locazione del segmento di linea relativo all'area rettangolare di definizione linea.

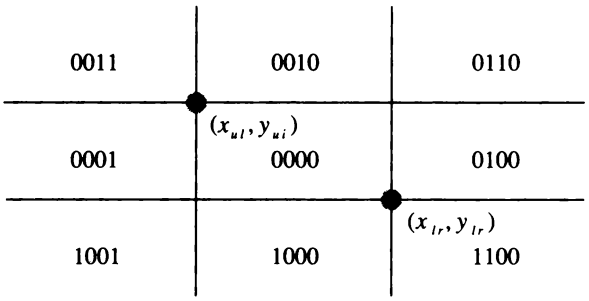


Figura 74. *Area rettangolare di definizione di linea utilizzando l'algoritmo di Sutherland-Cohen.*

L'algoritmo impiega una scorciatoia di calcolo per determinare la locazione relativa del segmento di linea. Ognuna delle nove aree viene rappresentata da un codice a 4 bit; ogni bit indica se l'area si trova al di sopra, al di sotto, a sinistra o a destra del rettangolo di definizione linea. La relazione delle estremità con il rettangolo viene quindi determinata rapidamente dalla combinazione binaria dei codici a 4 bit.

Se l'OR logico dei due codici è 0 (cioè, se il codice a 4 bit di entrambe le estremità è 0) entrambe le estremità si trovano all'interno del rettangolo, e non sarà necessario definire la linea. Se l'AND logico dei due codici a 4 bit non è zero, entrambe le estremità cadono sullo stesso lato del rettangolo, e l'intera linea viene definita. Questi test possono essere eseguiti rapidamente, in quanto entrambe le operazioni binarie AND e OR pssono essere implementate in singole istruzioni in linguaggio macchina 80x86.

Se l'OR logico dei codici a 4 bit delle estremità è diverso da zero ma l'AND logico è 0, allora il segmento di linea viene definito a ridosso di uno dei bordi del rettangolo. I valori dei codici a 4 bit determinano a quel punto quale bordo viene utilizzato. Il punto d'intersezione risultante diventa un nuovo punto terminale del segmento di linea.

Questo procedimento viene ripetuto per il nuovo segmento di linea. I codici a 4 bit vengono ricalcolati, il confronto logico viene nuovamente eseguito e, se necessario, viene calcolato un nuovo punto terminale. Dal momento che un rettangolo ha quattro lati, l'algoritmo richiede almeno quattro iterazioni.

La routine *Clip()* del Listato 71 è una dimostrazione in linguaggio C dell'algoritmo di Sutherland-Cohen. Il blocco *while* si ripete quando nessuna delle due condizioni di estremità (*Inside* o *Outside*) è vera. I codici a 4 bit vengono utilizzati per determinare quale dei quattro lati del rettangolo viene usato dal calcolo di definizione linea. Il punto di intersezione tra il segmento di linea e il lato del rettangolo diventa una nuova estremità. In fondo al blocco *while viene calcolato il codice a 4 bit della nuova estremità, e viene ripetuto il loop.*

```
/* Listato 71 */

struct        EndpointStruct        /* estremità per linea definita */
{
  int         x1,y1;
  int         x2,y2;
};

struct        RegionStruct          /* area rettangolare di definizione */
{
  int         Xul;
  int         Yul;
  int         Xlr;
  int         Ylr;
};


union OutcodeUnion   /* codici esterni rappresentati come campi di bit */
{
  struct
  {
    unsigned code0: 1;          /* x < Xul */
    unsigned code1: 1;          /* y < Yul */
    unsigned code2: 1;          /* x > Xlr */
    unsigned code3: 1;          /* y > Ylr */
              }
        ocs;
```

```
   int         outcodes;
};

#define        X1      ep->x1
#define        Y1      ep->y1
#define        X2      ep->x2
#define        Y2      ep->y2
#define        XUL     r>-Xul
#define        YUL     r>-Yul
#define        XLR     r>-Xlr
#define        YLR     r>-Ylr

Clip(ep,r)
struct         EndpointStruct     *ep;
struct         RegionStruct       *r;
{
               union OutcodeUnion     ocu1,ocu2;
               int     Inside;
               int     Outside;


               /* inizializza codici a 4 bit */

               SetOutcodes( &ocu1, r, X1, Y1 );  /* codici a 4 bit */
               SetOutcodes( &ocu2, r, X2, Y2 );  /*  iniziali /*

               Inside  = ((ocu1.outcodes | ocu2.outcodes) == 0);
               Outside = ((ocu1.outcodes & ocu2.outcodes) != 0);

               while (!Outside && !Inside)
               {
                 if (ocu1.outcodes==0)      /* scambia estremità se */
                 {                  /*  necessario in modo che (x1,y1)  */
                                    /*  debba essere definito /*
                   Swap( &X1, &X2 );
                   Swap( &Y1, &Y2 );
                   Swap( &ocu1, &ocu2 );
                 }

                 if (ocu1.ocs.code0)        /* definisce sinistra */
                 {
                   Y1 += (long)(Y2-Y1)*(XUL-X1)/(X2-X1);
                   X1 = XUL;
                 }

                 else if (ocu1.ocs.code1)   /* definisce sopra */
                 {
                   X1 += (long)(X2-X1)*(YUL-Y1)/(Y2-Y1);
                   Y1 = YUL;
                 }

                 else if (ocu1.ocs.code2)   /* definisce destra */
                 {
                   Y1 += (long)(Y2-Y1)*(XLR-X1)/(X2-X1);
                   X1 = XLR;
                 }

                 else if (ocu1.ocs.code3)   /* definisce sotto */
```

```
                {
                  X1 += (long)(X2-X1)*(YLR-Y1)/(Y2-Y1);
                  Y1 = YLR;
                }

                SetOutcodes( &ocu1, r, X1, Y1 );/* aggiorna per (x1,y1) */

                Inside  = ((ocu1.outcodes | ocu2.outcodes) == 0)
                                                ; /* aggiorna */
                Outside = ((ocu1.outcodes & ocu2.outcodes) != 0)
                                                ; /*  codici a 4bit */
              }

              return( Inside );


SetOutcodes( u, r, x, y )
union OutcodeUnion                  *u;
struct RegionStruct                 *r;
int     x,y;
{
              u->outcodes = 0;
              u->ocs.code0 = (x< XUL);
              u->ocs.code1 = (y< YUL);
              u->ocs.code2 = (x> XLR);
              u->ocs.code3 = (y> YLR);
}


Swap( pa, pb )
int    *pa,*pb;
{
              int               t;

              t = *pa;
              *pa = *pb;
              *pb = t;
}
```

Listato 71. *Un'implementazione dell'algoritmo di definizione linea di Sutherland-Cohen.*

Un programma potrebbe richiamare *Clip ()* prima di tracciare una linea con una veloce routine di base come *Line()*. Se prestate attenzione nel definire i valori XUL, YUL, XLR e YLR come variabili piuttosto che come costanti, potete utilizzare *Clip()* in qualsiasi modalità video. Inoltre, la definizione di linea non deve essere necessariamente limitata alla definizione di linee con i confini di RAM disponibile nel buffer del video. Potreste ad esempio voler definire un'area rettangolare arbitraria nel buffer del video e definire linee su quest'area. Un'interfaccia grafica di visualizzazione di buon livello supporta la definizione all'interno di queste aree arbitrarie.

7
Cerchi ed ellissi

Cerchi e calcolo in scala dei pixel

Un algoritmo per il tracciamento di un'ellisse
Conversione a scansione di un'ellisse
Un algoritmo incrementale
Una tipica implementazione
Problemi e trabocchetti
Precisione

Ottimizzazione

Definizione di ellissi

Cerchi reali

I cerchi e le ellissi sono probabilmente gli elementi grafici più diffusi dopo le linee rette. Il presente capitolo descrive le tecniche per la visualizzazione dei cerchi, delle ellissi e degli archi impiegando hardware di visualizzazione IBM. Queste tecniche sono analoghe agli algoritmi e agli esempi di programmazione per la visualizzazione di linee rette (descritti nel Capitolo 6). Sebbene una routine di tracciamento ellissi sia in qualche modo più complessa di una routine per il tracciamento di linee rette, il progetto algoritmico e le tecniche di programmazione sono simili.

Cerchi e calcolo in scala dei pixel

L'unico modo per tracciare un cerchio sui sistemi di visualizzazione IBM trattati in questo libro è quello di calcolare e di tracciare un'ellisse. La ragione di ciò è che la scala orizzontale nella quale vengono rappresentati i pixel si differenzia dalla scala verticale nella maggior parte delle modalità grafiche (Capitolo 4). Se visualizzate un "cerchio" i cui pixel vengono calcolati senza misurazione in scala, ciò che apparirà sullo schermo sarà un'ellisse. Ad esempio, La Figura 75a mostra un "cerchio" con un raggio di 100 pixel sia orizzontalmente sia verticalmente come viene visualizzato in una modalità grafica 640 per 200.

A causa del problema della misurazione in scala dei pixel, il tracciamento di un cerchio sullo schermo richiede che voi calcoliate i pixel che corrispondono ad un'ellisse matematica. In altre parole, per tracciare un cerchio che assomigli veramente ad un cerchio, dovrete calcolare un'ellisse i cui assi principale e secondario siano nello stesso rapporto del fattore di scala della coordinata di pixel. Sullo schermo, un'ellisse di questo tipo appare circolare (vedere Figura 71b). Per questo motivo, il presente capitolo si focalizza su un algoritmo reale per il tracciamento delle ellissi.

Un algoritmo per il tracciamento di un'ellisse

Conversione a scansione di un'ellisse

Per calcolare le coordinate x e y di tutti i pixel che rappresentano una data ellisse potete utilizzare la formula algebrica dell'ellisse. Come nel caso della conversione a scansione di una linea retta, molte di queste coordinate di pixel approssimeranno solamente i valori effettivi, e la figura risultante sarà dentellata. Questo effetto è particolarmente evidente quando si visualizza un'ellisse molto stretta (vedere Figura 76), ma nella maggior parte dei casi questo effetto secondario risulta accettabile.

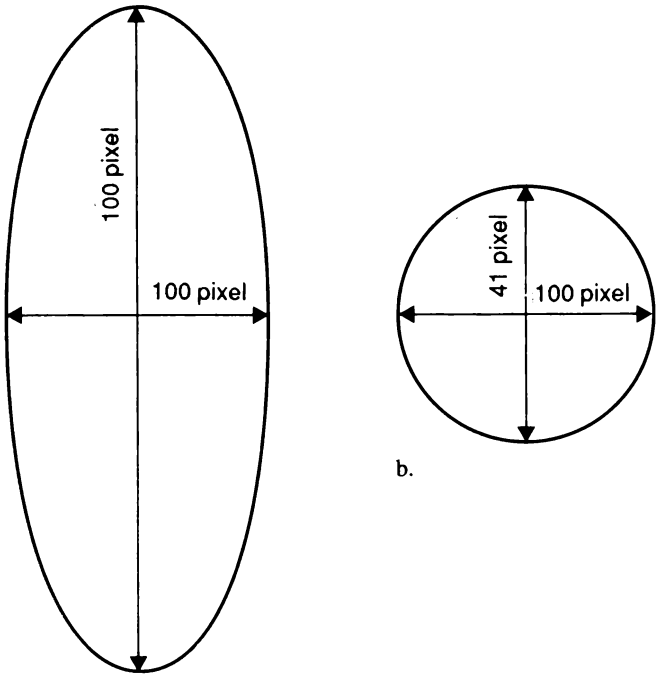


Figura 75. *Nella Figura 75a, un cerchio matematico con un raggio di 100 pixel appare ellittico nella modalità a 2 colori 640 per 200. Nella Figura 75b, un'ellisse i cui assi sono stati correttamente misurati in scala appare circolare quando viene visualizzata in questa modalità*

Per convertire a scansione e visualizzare un'ellisse potete usare l'equazione dell'ellisse:

$$(x-xc)^2/a^2 + (y-yc)^2/b^2 = 1$$

Questa equazione descrive un'ellisse con centro a (*xc,xy*) con gli assi principale e secondario *a* e *b* paralleli agli assi *x* e *y*.

Figura 76. *Un'ellisse stretta appare dentellata quando viene convertita a scansione.*

Tuttavia, le operazioni di servizio per i calcoli di tracciamento ellissi richieste dalla soluzione di questa equazione, come riportato nel Listato 72, sono parecchie. Le operazioni di moltiplicazione, divisione e di radice quadrata per determinare le coordinate di ogni pixel richiedono molto tempo. Un approccio migliore è quello di calcolare le coordinate di pixel incrementalmente in un modo analogo a quello utilizzato nell'algoritmo per il tracciamento linee del Capitolo 6.

```
/* Listato 72 */

Ellipse( xc, yc, a0, b0 )            /* uso dell'equazione di ellisse */
int           xc,yc;                 /* centro dell'ellisse */
int           a0,b0;                 /* assi principale e secondario */
{

              double x = 0;
              double y = b0;
              double Bsquared = (double)b0 * (double)b0;
              double Asquared = (double)a0 * (double)a0;
              double sqrt();


              do                     /* esegue mentre dy/dx> = -1 */
              {
                y = sqrt( Bsquared - ((Bsquared/Asquared) * x * x) );
                Set4Pixels( (int)x, (int)y, xc, yc, PixelValue );
                ++x;
              }
              while ( (x<=a0) && (Bsquared*x > Asquared*y) );
              while (y> = 0)         /* esegue mentre dy/dx < -1 */
              {
                x = sqrt( Asquared - ((Asquared/Bsquared) * y*y) );
                Set4Pixels( (int)x, (int)y, xc, yc, PixelValue );
                --y;
              }
}
5Set4Pixels( x, y, xc, yc, n )   /* imposta pixel in 4 quadranti per
                                    simmetria */
int           x,y;
int           xc,yc;
int           n;
{
              SPFunc(xc+x,yc+y,n);
              SPFunc(xc-x,yc+y,n);
              SPFunc(xc+x,yc-y,n);
              SPFunc(xc-x,yc-y,n);
}
```

Listato 72. *Tracciamento di un' ellisse usando l' equazione di ellisse.*

Un algoritmo incrementale

Il procedimento per ricavare un algoritmo incrementale per il tracciamento di ellissi assomiglia a quello per ricavare l'algoritmo di tracciamento delle linee di Bresenham. L'algoritmo di tracciamento ellissi traccia un'ellisse pixel per pixel. Dopo aver tracciato ogni pixel, l'algoritmo seleziona il pixel successivo determinando quale dei due pixel adiacenti al pixel corrente sia più vicino all'ellisse effettiva.

La creazione di un algoritmo di tracciamento ellissi è più semplice nel caso di un'ellisse che ha il centro all'origine del sistema di coordinate e gli assi principale e secondario giacenti sugli assi *x* e *y* (vedere Figura 74). L'equazione di un'ellisse di questo tipo è

$$b^2x^2 + a^2y^2 - a^2b^2 = 0$$

Dal momento che l'ellisse è simmetrica rispetto ad entrambi gli assi *x* e *y*, dovete solo ricavare un algoritmo per il tracciamento di uno dei suoi quadranti. La routine può quindi determinare per simmetria le coordinate di pixel degli altri tre quadranti.

L'algoritmo presentato qui è conosciuto come "algoritmo del punto di mezzo". Esso traccia iterativamente un'ellisse, pixel per pixel. Per ogni pixel tracciato, l'algoritmo seleziona quale dei pixel adiacenti approssima maggiormente l'ellisse, calcolando se il punto a metà distanza tra i pixel si trovi all'interno o all'esterno dell'ellisse (vedere Figura 78).

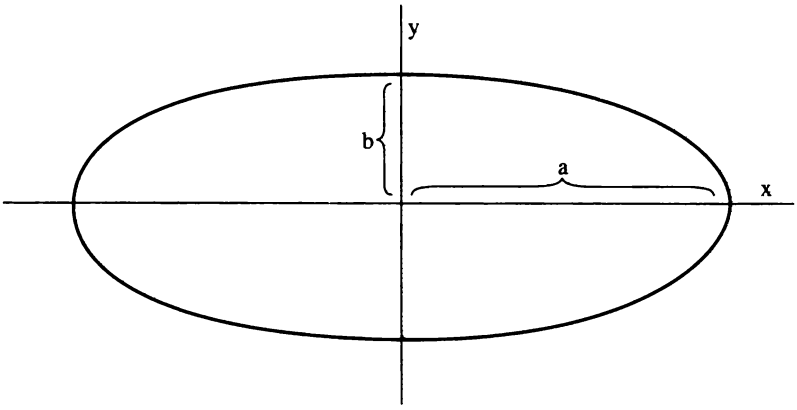


Figura 77. *Un' ellisse centrata sull' origine del sistema di coordinata.*

Per determinare quale pixel approssimi in modo migliore l'ellisse reale, l'algoritmo usa il valore dell'equazione dell'ellisse nel punto di mezzo tra i pixel. Se il valore è 0, il punto di mezzo ricade sull'ellisse. Se il valore è negativo, allora il punto di mezzo cade all'interno dell'ellisse; se il valore è positivo, il punto di mezzo cade all'esterno dell'el-

lisse. Di conseguenza, l'algoritmo può scegliere quale dei due pixel si avvicina di più all'ellisse reale esaminando il segno del valore.

Una complicazione sorge nella determinazione di quale coppia di pixel adiacenti verificare ad ogni fase dell'iterazione. Ciò dipende da dy/dx, la pendenza della tangente all'ellisse (vedere Figura 79). Quando dy/dx è maggiore di -1, l'algoritmo sceglie tra due pixel orientati verticalmente (vedere Figura 80a). Quando dy/dx è inferiore a -1, la scelta avviene tra due pixel orientati orizzontalmente (vedere Figura 80b).

Quando dy/dx è maggiore di -1, l'algoritmo determina iterativamente, per ogni pixel tracciato, se il pixel adiacente A o B approssima maggiormente l'ellisse reale. Ciò viene realizzato decidendo se il punto di mezzo tra A e B cade internamente o esternamente alIIl'ellisse esatta. Nella Figura 80a, il pixel selezionato nell'iterazione precedente si trova a (x_{i-1},y_{i-1}). Il punto di mezzo tra A e B è quindi $(x_{i-1}+1,y_{i-1}-1/2)$.

L'algoritmo sceglie tra il pixel A e il pixel B esaminando il segno del valore dell'equazione dell'ellisse nel punto di mezzo:

$$d = b^2(x_{i-1}+1)^2 + a^2(y_{i-1}-1/2)^2 - a^2b^2$$

La variabile d, che rappresenta il valore della funzione nel punto di mezzo, è la variabile decisionale dell'algoritmo. Come nell'algoritmo di tracciamento linea di Bresenham, la chiave della velocità di esecuzione di questo algoritmo è rappresentata dal fatto che può calcolare iterativamente d sulla base del suo valore ad ogni fase precedente nell'iterazione. La differenza tra il valore corrente di d ed il suo precedente valore è

$$\begin{aligned} d_i-d_i-1 &= [b^2(x_{i-1}+1)^2 + a^2(y_{i-1}-1/2)^2 - a^2b^2] - \\ &\quad [b^2(x_{i-1})^2 + a^2(y_{i-1}-1/2)^2 - a^2b^2 \\ &= b^2(x_{i-1}+1) \\ &= 2b^2x_{i-1}+b^2 \end{aligned}$$

A questo punto, trovare la differenza tra d_i e d_{i-1} (cioè dx) richiede ancora la moltiplicazione del valore precedente di x per una costante. Potete evitare questa moltiplicazione, però, calcolando dx, oltre che d, in modo incrementale; in pratica, aggiungendo $2b^2$ a dx ad ogni fase dell'iterazione.

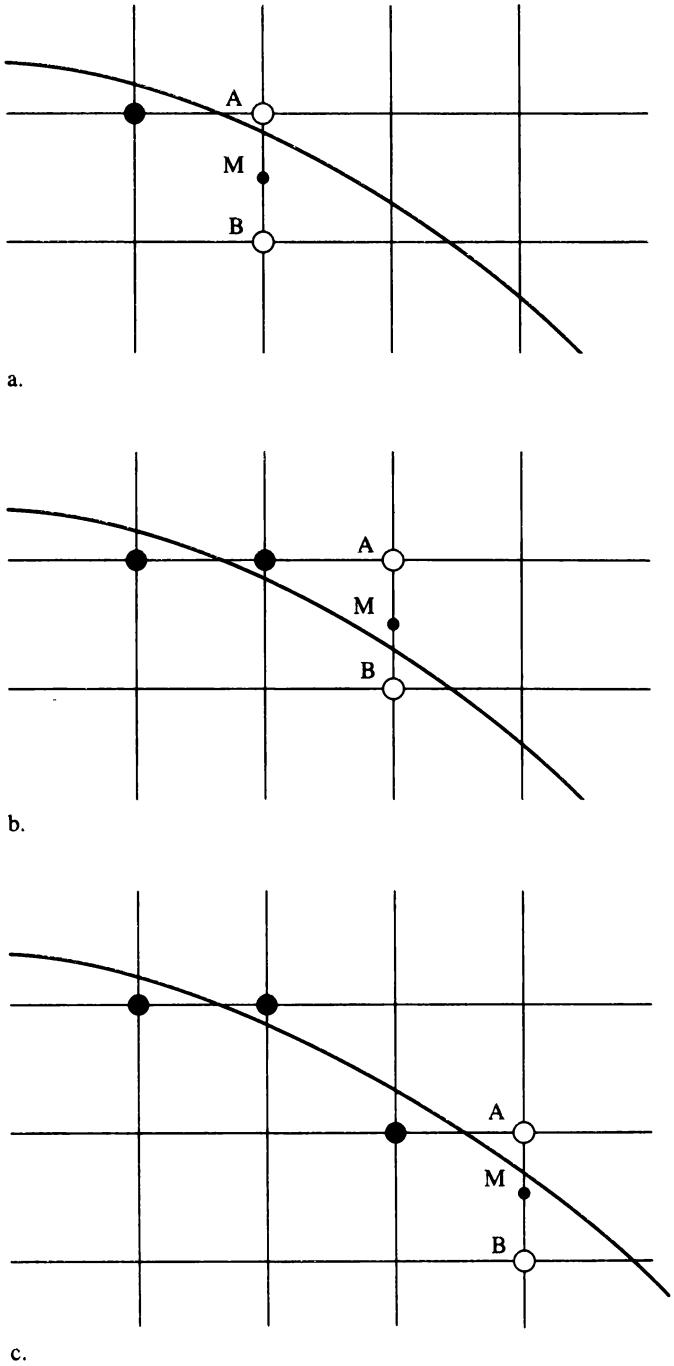


Figura 78. *Tre iterazioni dell'algoritmo del punto di mezzo. Dopo aver tracciato il pixel evidenziato con un grosso punto nero nell'illustrazione 78a, l'algoritmo sceglie di tracciare o il pixel* A *o il pixel* B *confrontando il punto* di mezzo M *con l'ellisse effettiva. Dal momento che* M *si trova all'interno dell'ellisse, sceglie il pixel* A. *Le illustrazioni 78b e 78c rappresentano le due iterazioni successive.*

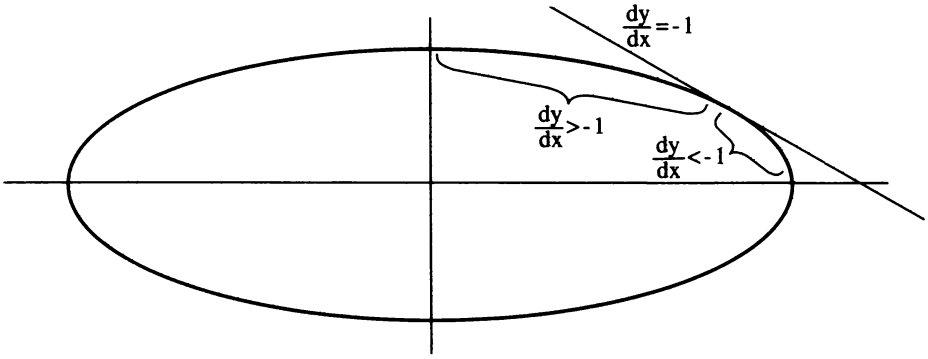
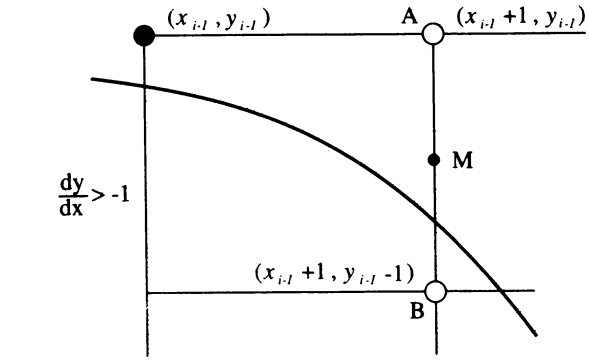
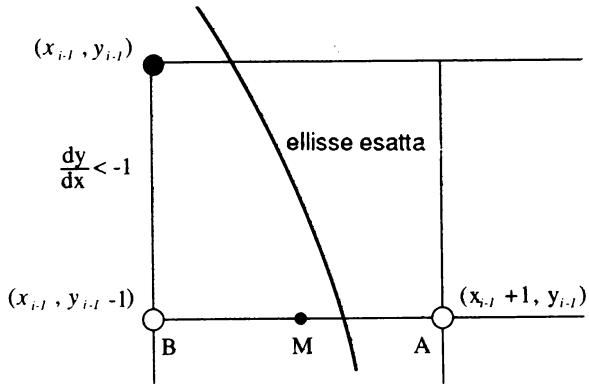


Figura 79. *La pendenza della tangente all' ellisse all' interno del primo quadrante.*



a.



b.

Figura 80 L'algoritmo del punto di mezzo sceglie tra A eB sostituendo x e y al punto di mezzo M nella formula dell'elisse e verificando il segno del risultato. Se il risultato è positivo, viene scelto il pixel B; se il risultato è negativo viene scelto il pixel A.

Se il pixel A si trova più vicino all'ellisse (cioè d_i > (di> 0)0), il valore appena calcolato di d_i può essere utilizzato come d_{i-1} nell'iterazione successiva. Se il pixel B si trova più vicino, però, d_i deve essere modificato in base alla fase discendente nella direzione y. In questo caso, deve essere calcolato il valore dell'equazione dell'ellisse nel punto di mezzo al di sotto del pixel B. Se $(x_{i-1},y_{i-1}-1/2)$ corrisponde al punto di mezzo tra i pixel A e B, allora $(x_{i-1},y_{i-1}-1/2)$ corrisponde al punto di mezzo al di sotto del pixel B, e dy è quindi

$$d_i-d_i-1 = [b^2(x_{i-1})^2 + a^2(y_{i-1}-1/2)^2 - a^2b^2] - [b^2(x_{i-1})^2 + a^2(y_{i-1}+1/2)^2 - a^2b^2]$$
$$= -2a^2y_{i-1}$$

Quando dy/dx è minore di -1, i pixel A e B sono pixel adiacenti orizzontali anziché verticali (vedere Figura 80b). I valori di dy e di dx sono quindi calcolati in modo diverso. Se viene scelto il pixel B, viene usato il punto di mezzo di coordinate $(x_{i-1}+1/2,y_{i-1}-1)$, in modo che l'incremento per d sia

$$d_i-d_i-1 = [b^2(x_{i-1}+1/2)^2 + a^2(y_{i-1}-1)^2 - a^2b^2] - [b^2(x_{i-1}+1/2)^2 + a^2(y_{i-1})^2 - a^2b^2]$$

$$=a^2(-2y_{i-1}+1)$$
$$=2a^2y_{i-1} + a^2$$

Inoltre, quando viene scelto il pixel A, d deve essere modificato per la fase della direzione verso destra:

$$d_i-d_{i-1} = [b^2(x_{i-1}+1/2)^2 + a^2(y_{i-1})^2 - a^2b^2] - [b^2(x_{i-1}-1/2)^2 + a^2(y_{i-1})^2 - a^2b^2]$$
$$=2b^2x_{i-1}$$

Queste derivazioni forniscono un modo per tracciare iterativamente un'ellisse, con semplici operazioni di addizione e sottrazione all'interno dei loop. L'analisi distingue tra il caso in cui dy/dx è maggiore di -1 ed il caso in cui dy/dx è minore di -1. Si determina quando dy/dx ha raggiunto -1 differenziando l'equazione dell'ellisse e impostando a -1 dy/dx.

$$(d/dx)(b^2x^2+a^2y^2-a^2b^2)=0$$

$$2b^2x+2a^2y(dy/dx)=0$$

$$dy/dx=-2b^2x/2a^2y$$

Di conseguenza, nel punto dove $dy/dx = -1$,

$$2b^2x = 2a^2y$$

Dal momento che l'algoritmo tiene già conto delle quantità $2b^2x$ e $2a^2y$ per calcolare i differenziali dx e dy, queste quantità possono essere usate per rilevare il punto in cui dy/dx raggiunge -1. L'algoritmo può quindi iniziare al punto $(0,b)$ sull'asse y e procedere in senso orario intorno all'ellisse fino a raggiungere $(a,0)$.

Inizialmente, la quantità dy/dx è maggiore di -1, e la scelta viene effettuata iterativamente tra i pixel orientati verticalmente (vedere Figura 80a). Quando dy/dx raggiunge -1, l'algoritmo sceglie tra i pixel orientati orizzontalmente (vedere Figura 80b) e continua in questo senso fino a raggiungere l'asse x.

L'unico calcolo restante avviene quando l'algoritmo raggiunge il pixel per il quale $dy/dx = -1$. A questo punto, un nuovo valore per d sarà già stato calcolato ($M_{vecchio}$ nella Figura 81) presumendo che il successivo punto di mezzo sarebbe stato tra due pixel orientati verticalmente. Di conseguenza, il valore di d deve essere modificato in modo che rispecchi il valore della funzione di ellisse al punto di mezzo tra due pixel orientati orizzontalmente (M_{nuovo} nella Figura 81). L'incremento di d (da $M_{vecchio}$ a M_{nuovo}) in questo caso è

$$\begin{aligned} d_i - d_i - 1 &= [b^2(x_{i-1}+1/2)^2 + a^2(y_{i-1}-1)^2 - a^2b^2] - \\ &\quad [b^2(x_{i-1}+1)^2 + a^2(y_{i-1}-1/2)^2 - a^2b^2] \\ &= b^2(-x_{i-1}-3/4 + a^2(-y_{i-1}+3/4) \\ &= 3(a^2-b^2)/4 - (b^2x_{i-1} + a^2y_{i-1}) \end{aligned}$$

Anche in questo caso, dal momento che l'algoritmo usa già le quantità $2b^2x$ e $2a^2y$, l'incremento di d a questo punto può essere calcolato da

$$d_i - d_i - 1 = 3(a^2-b)^2/4 - (2b^2x_{i-1}+2a^2y_{i-1})/2$$

L'aggiunta di questo valore a d al punto in cui $dy/dx = -1$ fornisce il nuovo valore di d.

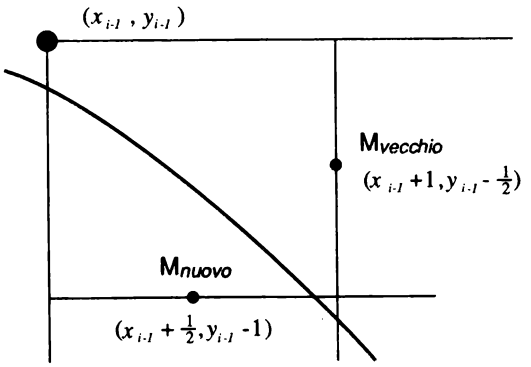


Figura 81. *Quando il valore di* dy/dx *raggiunge -1, viene selezionato un nuovo punto di mezzo (M_{nuovo}), e* d, *che è già stato calcolato per $M_{vecchio}$, viene modificato per rispecchiare il valore dell'equazione dell'ellisse a M_{nuovo}.*

Una tipica implementazione

La routine in C del Listato 73 è veloce ed efficiente in quanto tutti i calcoli di variabile decisionale all'interno dei loop iterativi interni sono stati ridotti in addizioni e sottrazioni. La routine elimina la moltiplicazione nei loop interni precalcolando i valori di a^2, b^2, $2a^2$ e $2b^2$. I valori iniziali delle variabili decisionali vengono calcolati presumendo che il primo pixel da tracciare sia a *(0,b)*. Di conseguenza, il valore iniziale di *d* viene calcolato per il punto di mezzo tra i pixel a *(1,b)* e *(1,b-1)*; cioè a *(1,b-1/2)*:

$$d = b^2(1)^2 + a^2(b-1/2)^2 - a^2b^2$$

$$= b^2 - a^2b + a^2/4$$

I valori iniziali di *dx* e di *dy* sono:

$$dx = 2b^2(x_0) = 0$$

e

$$dy = 2a^2(y_0) = 2a^2b$$

La routine *Ellipse()* segue strettamente l'algoritmo. Per prima cosa traccia tutti i pixel tra *(0,b)* ed il punto dove *dy/dx* diventa -1, quindi aggiorna *d* come nella Figura 81. La selezione iterativa di pixel continua fino a che la routine non raggiunge l'asse *x*. La routine richiama la funzione *Set4Pixels()* per duplicare ogni pixel in ognuno dei quattro quadranti dell'ellisse. *Set4Pixels()* converte anche le coordinate di ogni pixel relative al centro effettivo dell'ellisse.

```
/* Listato 73 */

Ellipse( xc, yc, a0, b0 )
int             xc,yc;                  /* centro dell'ellisse */
int             a0,b0;                  /* semiasse */
{
                int     x = 0;
                int     y = b0;

                long    a = a0;                     /* usa precisione a 32 bit */
                long    b = b0;

                long    Asquared = a * a;           /* inizializza valori */
                                                    /*  all'esterno dei loop */
                long    TwoAsquared = 2 * Asquared;
                long    Bsquared = b * b;
                long    TwoBsquared = 2 * Bsquared;

                long    d;
                long    dx,dy;
```

```
            d = Bsquared - Asquared*b + Asquared/4L;
            dx = 0;
            dy = TwoAsquared * b;

            while (dx <dy)
            {
              Set4Pixels( x, y, xc, yc, PixelValue );

              if (d>0L)
              {
                --y;
                dy -= TwoAsquared;
                d -= dy;
              }

              ++x;
              dx += TwoBsquared;
              d += Bsquared + dx;
            }


            d += (3L*(Asquared-Bsquared)/2L - (dx+dy)) / 2L;

            while (y>=0)
            {
              Set4Pixels( x, y, xc, yc, PixelValue );

              if (d<0L)

              {
                ++x;
                dx += TwoBsquared;
                d += dx;
              }

              --y;
              dy -= TwoAsquared;
              d += Asquared - dy;
            }
}

Set4Pixels( x, y, xc, yc, n )        /* imposta pixel per simmetria nei 4 */
                                     /*  quadranti */
int         x,y;
int         xc,yc;
int         n;
{
            SetPixel( xc+x, yc+y, n );
            SetPixel( xc-x, yc+y, n );
            SetPixel( xc+x, yc-y, n );
            SetPixel( xc-x, yc-y, n );
}
```

Listato 73. *Un' implementazione in un linguaggio ad alto livello dell' algoritmo del punto di mezzo.*

Problemi e trabocchetti

Un problema difficile che incontrerete è rappresentato dal fatto che le ellissi strette appaiono in qualche modo spigolose piuttosto che ellittiche quando vengono convertite in scansione. Quando un'ellisse è piccola e vengono utilizzati relativamente pochi pixel per visualizzarla, l'approssimazione migliore generata dall'algoritmo potrebbe apparire come un poligono.

Sebbene in queste situazioni sia possibile riprogettare l'algoritmo di tracciamento ellisse per tracciare ellissi con contorni più "spessi" o più "sottili", una soluzione migliore è rappresentata dal visualizzare le ellissi con una maggiore risoluzione. Le ellissi con contorni più sottili assumono un aspetto di gran lunga migliore con una risoluzione 640 per 480 che con una risoluzione 320 per 200.

Un problema correlato è che ellissi molto eccentriche possono essere tracciate in modo poco accurato nei punti dove la curva è maggiormente stretta. Ciò avviene quando il punto dove cade $dy,dx = -1$ è quasi adiacente o all'asse x o all'asse y. Anche in questo caso, potete modificare l'algoritmo in modo che si adatti alla situazione, ma se la vostra applicazione richiede rappresentazioni precise di ellissi molto sottili, una soluzione migliore è quella di visualizzarle a risoluzioni maggiori.

Un'ulteriore considerazione riguarda le ellissi "degenerate" per le quali la lunghezza dell'asse principale o di quello secondario è 0 (cioè, a=0 o b=0). Dal momento che o dy o dx avranno valore 0 in questa situazione, le routine iterative non verranno concluse in modo corretto. In questi casi, o effettuate i test per le condizioni particolari prima di eseguire i loop (e tracciate la linea retta appropriata) o modificate le condizioni di conclusione dei loop.

Precisione

Come avviene con l'algoritmo per il tracciamento linee di Bresenham, l'algoritmo del punto di mezzo tenta di ridurre la distanza verticale o orizzontale tra l'ellisse e i pixel che seleziona. Ciò avviene in modo più veloce rispetto alla riduzione della distanza tra ogni pixel e il punto più vicino ad esso sull'ellisse, ma se esaminate attentamente i risultati, potreste scoprire che raramente il pixel selezionato dall'algoritmo del punto di mezzo non è quello più vicino all'ellisse. Ciononostante, la precisione dell'algoritmo del punto di mezzo nella selezione dei pixel migliori per rappresentare l'ellisse è sufficiente per quasi tutte le applicazioni.

Sebbene il codice sorgente del Listato 73 rappresenti una semplice implementazione dell'algoritmo, dovete ricordare alcuni dettagli se pensate di modificare il codice o di tradurlo in un altro linguaggio. E' importante calcolare tutte le variabili decisionali come numeri interi a 32 bit. Dal momento che questi valori comportano l'elevazione al quadrato delle coordinate di pixel, 16 bit non sono adeguati per conservare la precisione.

Un altro dettaglio da ricordare è che questa routine può tracciare due volte gli stessi pixel. Questo avviene grazie alla simmetria a quattro lati dell'ellisse. Ad esempio, i pixel a (+/-*a*,0) e (0,+/-*b*) vengono aggiornati due volte da *Set4Pixels()* nel Listato 73. Questo diventa un problema quando usate la routine per effettuare l'XOR dei pixel nel buffer del video. Se eseguite due volte un'operazione XOR su questi pixel, essi scompariranno. Per evitare questo, o verificate i casi particolari in *Set4Pixels()* (Listato 74) o modificate *Ellipse()* in modo che tracci separatamente questi pixel.

```
/* Listato 74 */

Set4Pixels( x, y, xc, yc, n )    /* evita la doppia impostazione dello */
                                 /*  stesso pixel */
int            x,y;
int            xc,yc;
int            n;
{
               if (x!=0)
               {
                 SetPixel( xc+x, yc+y, n );
                 SetPixel( xc-x, yc+y, n );
                 if (y!=0)
                 {
                   SetPixel( xc+x, yc-y, n );
                   SetPixel( xc-x, yc-y, n );
                 }
               }
               else
               {
                 SetPixel( xc, yc+y, n );
                 if (y!=0)
                   SetPixel( xc, yc-y, n );
               }
}
```

Listato 74. *Una versione modificata di Set4Pixels che evita di aggiornare due volte lo stesso pixel.*

Ottimizzazione

Per molte applicazioni, un'implementazione in un linguaggio ad alto livello come quella del Listato 73 risulta sufficientemente veloce. La parte più lenta della versione ad alto livello di *Ellipse()* è rappresentata dalle ripetute chiamate alla routine di impostazione pixel, che ricalcola gli indirizzi di pixel ad ogni iterazione. Scrivendo *Ellipse()* in linguaggio Assembler, potete calcolare gli indirizzi di pixel molto più efficientemente. La risultante routine in linguaggio Assembler risulta circa tre volte più veloce del suo equivalente in linguaggio ad alto livello.

Il Listato 75 rappresenta una tipica implementazione in linguaggio Assembler, in questo caso per EGA. Si noti che la routine *Set4Pixels* mantiene una serie di quattro off-

set di buffer e di maschere di bit al posto delle coordinate (*x,y*) per i quattro pixel da aggiornare. Quando *Set4Pixels* incrementa una coordinata *x* di pixel, ruota una maschera di bit nella direzione corretta. Le coordinate *y* vengono incrementate aggiungendo il numero di byte di ogni linea di pixel all'offset del buffer (questa è la stessa tecnica utilizzata nelle routine di tracciamento linee del Capitolo 6). Questo metodo di indirizzamento di buffer di visualizzazione è molto più veloce della chiamata ad una funzione *Set4Pixel()* per ogni pixel dell'ellisse.

```
                TITLE   'Listato 75'
                NAME    Ellipse10
                PAGE    55,132

;
; Nome:         Ellipse10
;
; Funzione:     Traccia un'ellisse nelle modalità grafiche originali EGA/VGA.
;
; Chiamante:    Microsoft C:
;
;                       azzera Ellipse10(xc,yc,a,b,n);
;
;                       int xc,yc; /* centro dell'ellisse */
;
;                       int a,b;   /* assi principale e secondario */
;
;                       int n;     /* valore di pixel */
;

ARGxc           EQU     word ptr [bp+4] ; indirizzamento di cornice di stack
ARGyc           EQU     word ptr [bp+6]
ARGa            EQU     word ptr [bp+8]
ARGb            EQU     word ptr [bp+10]
ARGn            EQU     byte ptr [bp+12]

ULAddr          EQU     word ptr [bp-2]
URAddr          EQU     word ptr [bp-4]
LLAddr          EQU     word ptr [bp-6]
LRAddr          EQU     word ptr [bp-8]
LMask           EQU     byte ptr [bp-10]
RMask           EQU     byte ptr [bp-12]

VARd            EQU     word ptr [bp-16]
VARdx           EQU     word ptr [bp-20]
VARdy           EQU     word ptr [bp-24]
Asquared        EQU     word ptr [bp-28]
Bsquared        EQU     word ptr [bp-32]
TwoAsquared     EQU     word ptr [bp-36]
TwoBsquared     EQU     word ptr [bp-40]
```

```
RMWbits        EQU    0              ; legge-modifica-scrive bit
BytesPerLine   EQU    80


_TEXT          SEGMENTbyte public 'CODE'
               ASSUME cs:_TEXT

               EXTRN  PixelAddr10:near

               PUBLIC _Ellipse10
_Ellipse10     PROC   near

               push   bp             ; mantiene registri del chiamante
               mov    bp,sp
               sub    sp,40          ; stabilisce cornice di stack
               push   si
               push   di

; imposta registro di modalità del controller grafico

               mov    dx,3CEh        ; DX := porta di I/O control. grafico
               mov    ax,0005h       ; AL := numero registro di modalità
                                     ; AH := modal. scrittura 0 (bit 0,1)
               out    dx,ax          ;       modalità di lettura 0 (bit 4)

; imposta registro di rotazione dati/selezione funzione

               mov    ah,RMWbits     ; AH := legge-modifica-scrive bit
               mov    al,3           ; AL := reg rotazione dati/selezione
                                     ;  funzione
               out    dx,ax

; imposta registri impostazione/azzeramento e abilitazione
; impostazione/azzeramento

               mov    ah,ARGn        ; AH := valore del pixel
               mov    al,0           ; AL := numero reg. impostaz./azzer.
               out    dx,ax

               mov    ax,0F01h       ; AH := valore di abilitazione
                                     ;  impostazione/azzeramento
                                     ;  (tutti i piani di bit abilitati)
               out    dx,ax          ; AL := numero reg abilitazione
                                     ;  impostazione/azzeramento

; costanti iniziali

               mov    ax,ARGa
               mul    ax
               mov    Asquared,ax
               mov    Asquared+2,dx     ; a^2
               shl    ax,1
               rcl    dx,1
               mov    TwoAsquared,ax
               mov    TwoAsquared+2,dx ; 2*a^2
```

```
            mov     ax,ARGb
            mul     ax
            mov     Bsquared,ax
            mov     Bsquared+2,dx         ; b^2
            shl     ax,1
            rcl     dx,1
            mov     TwoBsquared,ax
            mov     TwoBsquared+2,dx      ; 2*b^2
;
; traccia pixel da (0,b) fino a dy/dx = -1
;

; indirizzo di buffer e mashera di bit iniziali

            mov     ax,BytesPer line
                                          ; AX := lunghezza di linea di
                                          ;  bufferdi visualizzazione
            mul     ARGb                  ; AX := offset di byte relativo di b
            mov     si,ax
            mov     di,ax

            mov     ax,ARGyc              ; AX := yc
            mov     bx,ARGxc              ; BX := xc
            call    PixelAddr10           ; AH := maschera di bit
                                          ; ES:BX -> buffer
                                          ; CL := # bit da spostare a sinistra
            mov     ah,1
            shl     ah,cl                 ; AH := maschera di bit per primo
                                          ;  pixel
            mov     LMask,ah
            mov     RMask,ah

            add     si,bx                 ; SI := offset di (0,b)
            mov     ULAddr,si
            mov     URAddr,si
            sub     bx,di                 ; AX := offset di (0,-b)
            mov     LLAddr,bx
            mov     LRAddr,bx

; variabili decisionali iniziali

            xor     ax,ax
            mov     VARdx,ax
            mov     VARdx+2,ax            ; dx = 0

            mov     ax,TwoAsquared
            mov     dx,TwoAsquared+2
            mov     cx,ARGb
            call    LongMultiply          ; esegue moltiplicazione 32 bit per
                                          ;  16 bit
            mov     VARdy,ax
            mov     VARdy+2,dx            ; dy = TwoAsquared * b

            mov     ax,Asquared
            mov     dx,Asquared+2         ; DX:AX = Asquared
```

```
            sar     dx,1
            rcr     ax,1
            sar     dx,1
            rcr     ax,1            ; DX:AX = Asquared/4

            add     ax,Bsquared
            adc     dx,Bsquared+2   ; DX:AX = Bsquared + Asquared/4
            mov     VARd,ax
            mov     VARd+2,dx

            mov     ax,Asquared
            mov     dx,Asquared+2
            mov     cx,ARGb
            call    LongMultiply    ; DX:AX = Asquared*b
            sub     VARd,ax
            sbb     VARd+2,dx       ; d = Bsquared - Asquared*b +
                                    ;  Asquared/4
; loop fino a che dy/dx> = -1

            mov     bx,ARGb         ; BX := coordinata y iniziale

            xor     cx,cx           ; CH := 0 (incremento y iniziale)
                                    ; CL := 0 (incremento x iniziale)
L10:        mov     ax,VARdx
            mov     dx,VARdx+2
            sub     ax,VARdy
            sbb     dx,VARdy+2
            jns     L20             ; salta se dx>=dy

            call    Set4Pixels

            mov     cx,1            ; CH := 0 (incremento y)
                                    ; CL := 1 (incremento x)
            cmp     VARd+2,0
            js      L11             ; salta se d>0

            mov     ch,1            ; incremento in direzione y
            dec     bx              ; decrementa coordinata y corrente

            mov     ax,VARdy
            mov     dx,VARdy+2
            sub     ax,TwoAsquared
            sbb     dx,TwoAsquared+2  ; DX:AX := dy - TwoAsquared
            mov     VARdy,ax
            mov     VARdy+2,dx      ; dy -= TwoAsquared

            sub     VARd,ax
            sbb     VARd+2,dx       ; d -= dy

L11:        mov     ax,VARdx
            mov     dx,VARdx+2
            add     ax,TwoBsquared
            adc     dx,TwoBsquared+2  ; DX:AX := dx + TwoBsquared
            mov     VARdx,ax
            mov     VARdx+2,dx      ; dx += TwoBsquared
```

```
            add     ax,Bsquared
            adc     dx,Bsquared+2   ; DX:AX := dx + Bsquared
            add     VARd,ax
            adc     VARd+2,dx       ; d += dx + Bsquared

            jmp     L10
;
; traccia pixel da (x,y) corrente fino a y<0
;

; indirizzo di buffer e maschera di bit iniziali

L20:        push    bx              ; mantiene coordinata y corrente
            push    cx              ; mantiene incrementi x e y

            mov     ax,Asquared
            mov     dx,Asquared+2

            sub     ax,Bsquared
            sbb     dx,Bsquared+2   ; DX:AX := Asquared-Bsquared

            mov     bx,ax
            mov     cx,dx           ; CX:BX := (Asquared-Bsquared)

            sar     dx,1
            rcr     ax,1            ; DX:AX := (Asquared-Bsquared)/2
            add     ax,bx
            adc     dx,cx           ; DX:AX := 3*(Asquared-Bsquared)/2

            sub     ax,VARdx
            sbb     dx,VARdx+2
            sub     ax,VARdy
            sbb     dx,VARdy+2      ; DX:AX := 3*(Asquared-Bsquared)/2 -
                                    ;  (dx+dy)

            sar     dx,1
            rcr     ax,1            ; DX:AX :=
                                    ;  ( 3*(Asquared-Bsquared)/2 -
                                    ;  (dx+dy) )/2
            add     VARd,ax
            adc     VARd+2,dx       ; aggiorna d

; loop fino a che y<0

            pop     cx              ; CH,CL := incrementi y e x
            pop     bx              ; BX := y

L21:        call    Set4Pixels

            mov     cx,100h         ; CH := 1 (incremento y)
                                    ; CL := 0 (incremento x)
            cmp     VARd+2,0
            jns     L22             ; salta se d>=0

            mov     cl,1            ; incremento in direzione x
```

```
            mov     ax,VARdx
            mov     dx,VARdx+2
            add     ax,TwoBsquared
            adc     dx,TwoBsquared+2 ; DX:AX := dx + TwoBsquared
            mov     VARdx,ax
            mov     VARdx+2,dx       ; dx += TwoBsquared

            add     VARd,ax
            adc     VARd+2,dx        ; d += dx

L22:        mov     ax,VARdy
            mov     dx,VARdy+2
            sub     ax,TwoAsquared
            sbb     dx,TwoAsquared+2 ; DX:AX := dy - TwoAsquared
            mov     VARdy,ax
            mov     VARdy+2,dx       ; dy -= TwoAsquared

            sub     ax,Asquared
            sbb     dx,Asquared+2    ; DX:AX := dy - Asquared
            sub     VARd,ax
            sbb     VARd+2,dx        ; d += Asquared - dy

            dec     bx               ; decrementa y
            jns     L21              ; loop se y>=0


; ripristina registri di default del controller grafico

Lexit:      mov     ax,0FF08h        ; maschera di bit di default
            mov     dx,3CEh
            out     dx,ax

            mov     ax,0003          ; selezione funzione di default
            out     dx,ax

            mov     ax,0001          ; abilitaz. impost./azzer. di default
            out     dx,ax

            pop     di               ; ripristina registri e ritorna
            pop     si
            mov     sp,bp
            pop     bp
            ret

_Ellipse10  ENDP


Set4Pixels  PROC    near             ; Richiama con:  CH := incremento y
                                     ;                (0, -1)
                                     ;                CL := incremento x
                                     ;                (0, 1)

            push    ax               ; mantiene questi registri
            push    bx
            push    dx
```

```
            mov     dx,3CEh         ; DX := porta del controller grafico

            xor     bx,bx           ; BX := 0
            test    ch,ch
            jz      L30             ; salta se incremento y= 0

            mov     bx,BytesPer line
                                    ; BX := incremento positivo
            neg     bx              ; BX := incremento negativo

L30:        mov     al,8            ; AL := numero reg maschera di bit

; pixel a (xc-x,yc+y) e (xc-x,yc-y)

            xor     si,si           ; SI := 0
            mov     ah,LMask

            rol     ah,cl           ; AH := maschera di bit ruotata
                                    ;  orizzontalmente
            rcl     si,1            ; SI := 1 se maschera di bit è stata
                                    ;  ruotata
            neg     si              ; SI := 0 o -1

            mov     di,si           ; SI,DI := incremento orizz. sinistro

            add     si,ULAddr       ; SI := indirizzo superiore sinistro
                                    ;  + incr oriz
            add     si,bx           ; SI := nuovo indirizzo superiore
                                    ;  sinistro
            add     di,LLAddr
            sub     di,bx           ; DI := nuovo indirizzo inferiore
                                    ;  sinistro

            mov     LMask,ah        ; aggiorna queste variabili
            mov     ULAddr,si
            mov     LLAddr,di

            out     dx,ax           ; aggiorna reg. di maschera di bit

            mov     ch,es:[si]      ; aggiorna pixel superiore sinistro
            mov     es:[si],ch
            mov     ch,es:[di]      ; aggiorna pixel inferiore sinistro
            mov     es:[di],ch


; pixel a (xc+x,yc+y) e (xc+x,yc-y)

            xor     si,si           ; SI := 0
            mov     ah,RMask

            ror     ah,cl           ; AH := maschera di bit ruotata
                                    ;  orizzontalmente
            rcl     si,1            ; SI := 1 se maschera di bit è stata
                                    ;  ruotata

            mov     di,si           ; SI,DI := incremento orizzon. destro
```

```
            add     si,URAddr       ; SI := indirizzo superiore destro +
                                    ;  incr oriz
            add     si,bx           ; SI := nuovo indirizzo superiore
                                    ;  destro
            add     di,LRAddr
            sub     di,bx           ; DI := nuovo indirizzo inferiore
                                    ;  destro

            mov     RMask,ah        ; aggiorna queste variabili
            mov     URAddr,si
            mov     LRAddr,di

            out     dx,ax           ; aggiorna reg. di maschera di bit

            mov     ch,es:[si]      ; aggiorna pixel superiore destro
            mov     es:[si],ch
            mov     ch,es:[di]      ; aggiorna pixel inferiore destro
            mov     es:[di],ch

            pop     dx              ; ripristina questi registri
            pop     bx
            pop     ax
            ret

Set4Pixels  ENDP

LongMultiply PROC   near            ; Chiamante:   DX = u1 (word più
                                    ;               significativa
                                    ;               di numero a 32 bit)
                                    ;              AX = u2 (word meno
                                    ;               significativa)
                                    ;              CX = v1 (numero a 16
                                    ;               bit)
                                    ; Ritorna:  DX:AX = risultato a 32
                                    ;               bit)

            push    ax              ; mantiene u2
            mov     ax,dx           ; AX :=
            mul     cx              ; AX :=word più significativa del
                                    ;  risultato
            xchg    ax,cx           ; AX := v1, CX :=word più signif.
            pop     dx              ; DX := u2
            mul     dx              ; AX :=word meno significativa del
                                    ;  risultato
                                    ; DX := riporto
            add     dx,cx           ; CX :=word più significativa del
                                    ;  risultato
            ret

LongMultiply ENDP

_TEXT       ENDS

            END
```

Listato 75. *Un' implementazione in linguaggio Assembler dell' algoritmo del punto di mezzo.*

La tecnica di ottimizzazione usata nel Capitolo 6 viene qui omessa. In pratica, non vale la pena ridurre gli accessi al buffer del video impostando più di un pixel per volta in ogni byte del buffer. Le operazioni di servizio necessarie per tener conto di quali byte contengono più di un pixel aggiornato sono maggiori del tempo risparmiato nel ridurre gli accessi al buffer del video. Inoltre, il codice risulta già abbastanza complicato.

Definizione di ellissi

Se definite un'ellisse all'interno di una finestra rettangolare, il risultato sarà un arco (vedere Figura 82). La sezione dove eseguire la definizione è quella della routine *Set4Pixels()*. Potete definire le coordinate di ogni pixel (*x,y*) a ridosso dei confini di finestra prima di chiamare *SetPixel()* per aggiornare il buffer del video.

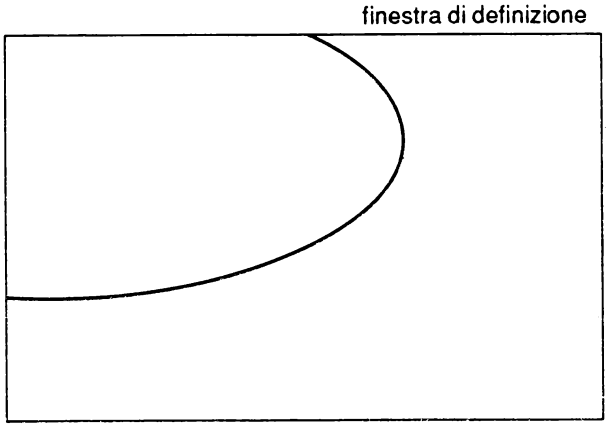


Figura 82. *La definizione di un'ellisse produce un arco.*

L'implementazione della definizione di ellisse in questo modo rallenta in qualche modo la routine di tracciamento ellissi. Se la vostra applicazione richiede raramente la definizione, considerate l'implementazione di due diverse versioni di *Set4Pixels()*, una che effettua la definizione e una che la omette. Prima di chiamare *Ellipse()*, potete confrontare i valori minimo e massimo di coordinata dei pixel dell'ellisse (*xc+/-a,yc+/-b*) con i confini della definizione per determinare se può essere tracciata senza definizione. Solo se la definizione è necessaria dovete utilizzare la versione più lenta comprendente la definizione di ellisse di *Set4Pixels()*.

Cerchi reali

Dopo aver implementato la routine di tracciamento ellisse, potete tracciare cerchi reali in tutte le modalità grafiche sui sistemi di visualizzazione dei PC e dei PS/2. Per visualizzare un cerchio, tracciate un'ellisse con gli assi principale e secondario calcolati in scala in proporzione alle risoluzioni orizzontale e verticale del vostro monitor. Il Listato 76 mostra come potreste effettuare questa operazione nella modalità grafica 640 per 350 su EGA.

Dal momento che il calcolo in scala varia con la modalità video, la stessa routine non può tracciare cerchi nelle diverse modalità video a meno che non contenga il calcolo in scala della coordinata di pixel in ogni modalità. La Figura 49 del Capitolo 4 contiene una tabella di fattori di scala di pixel per tutte le modalità grafiche.

```
/* Listato 76 */

Circle10( xc, yc, xr, yr, n)                /* cerchi nella modalità
                                               a 16 colori 640x350 */
int           xc,yc;                        /* centro del cerchio */
int           xr,yr;                        /* punto sulla circonferenza */
int           n;                            /* valore di pixel */
{
              double x,y;
              double sqrt();
              double Scale10 = 1.37;        /* fattore di scala di pixel */
              int    a,b;

              x = xr - xc;                  /* converte centro dell'ellisse */
              y = (yr - yc) * Scale10;      /*  all'origine */


              a = sqrt( x*x + y*y );        /* calcola assi principale
                                               e secondario */
              b = a / Scale10;

              Ellipse10( xc, yc, a, b, n); /* traccia ellisse */
}
```

Listato 76. *Uso del calcolo in scala della coordinata di pixel per visualizzare un cerchio nella modalità a 16 colori 640 per 350.*

8
Riempimento di aree

Cos'è un'area?
Pixel interni e di cornice - Collegamento

Riempimenti semplici con linee orizzontali

Tre algoritmi di riempimento aree
Semplice riempimento ripetitivo
Riempimento di linee adiacenti
Riempimento di cornice

Confronto degli algoritmi

Il presente capitolo descrive diversi metodi per il riempimento di un'area del buffer del video con una maschera di pixel. Le tecniche di riempimento aree vengono utilizzate in molti settori della programmazione grafica per computer, tra cui la manipolazione del colore, l'ombreggiatura e la rappresentazione tridimensionale degli oggetti, oltre che nelle applicazioni come l'elaborazione delle immagini, la trasmissione dei dati di immagini e l'animazione su computer.

Questo capitolo contiene codici sorgente operativi per tre algoritmi di riempimento aree, ma quanto esposto non è da intendersi nel modo più assoluto come una spiegazione esaustiva sull'argomento. Questi algoritmi e queste implementazioni hanno lo scopo di rappresentare modelli operativi da sperimentare, modificare ed ottimizzare per le vostre applicazioni.

Cos'è un'area?

Un'area è un gruppo di pixel collegati all'interno del buffer del video, delimitata da un qualche tipo di confine. Potete pensare ad un'area del buffer del video come se fosse composta da una parte interna e da una cornice. Per comprendere come gli algoritmi di questo capitolo vengono implementati, tuttavia, è importante considerare come una regione possa essere definita chiaramente in termini di valori di pixel e di geometria di pixel all'interno del buffer del video.

Pixel interni e di cornice

Nel corso di questo capitolo un'area viene considerata circondata da pixel i cui valori li distinguono dai pixel interni. Potreste presupporre, ad esempio, che tutti i pixel interni hanno lo stesso valore, nel qual caso un pixel di cornice è semplicemente un qualsiasi pixel il cui valore è diverso dai valori dei pixel interni (vedere Figura 83a). Potreste anche assegnare una gamma di valori di pixel ammessi sia ai pixel di cornice sia a quelli interni. Gli algoritmi di questo capitolo sono fedeli alla convenzione che tutti i pixel della cornice hanno un valore specificato e che i pixel interni possono avere un qualsiasi altro valore (vedere Figura 83b).

In molte applicazioni, risulta pratico utilizzare una gamma di coordinate di pixel per definire per intero o parzialmente la cornice di un'area. La definizione di un "pixel di cornice" può quindi essere allargata per includere i pixel al di fuori di una gamma predeterminata di coordinate (x,y). In questo modo un'area può essere delimitata dai confini del buffer di schermo o da una finestra software, oltre che dai pixel di un valore o di una gamma di valori predeterminati.

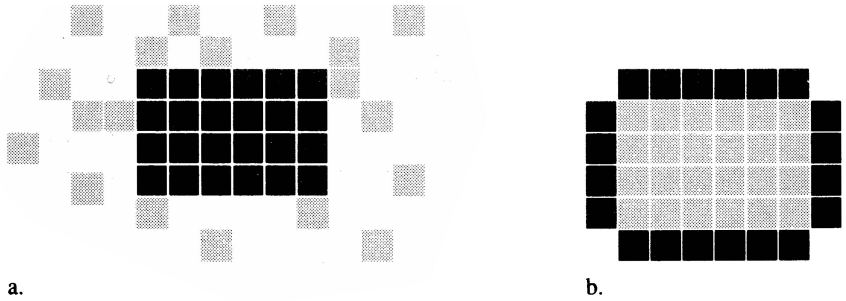


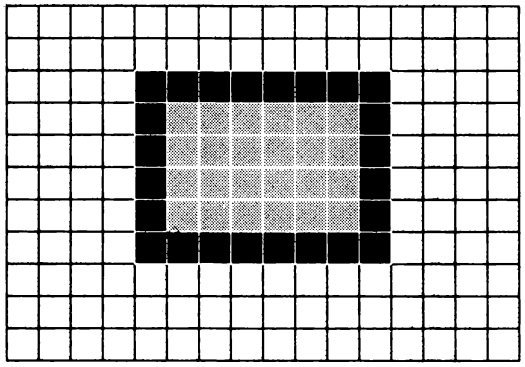
Figura 83. *Nella Figura 83a, un'area viene definita dai pixel interni di un dato valore. Nella Figura 83b, un'area viene definita dai pixel di cornicce di un dato valore.*

Collegamento

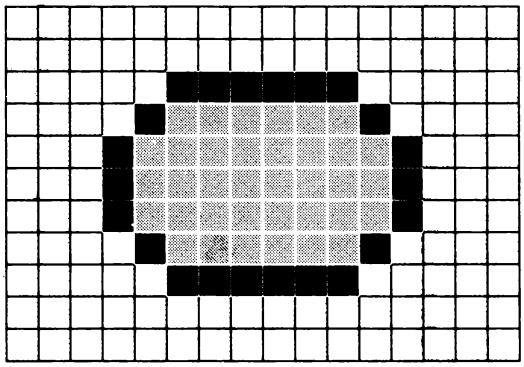
Per distinguere i pixel di cornice dai pixel interni, dovete anche specificare il modo in cui i pixel vengono collegati. Se permettete che i pixel interni si colleghino diagonalmente oltre che ortogonalmente (orizzontalmente e verticalmente), dovete presumere che i pixel di cornice che circondano l'area siano sempre collegati ortogonalmente (vedere Figura 84a). Per contro, se permettete che i pixel di cornice si colleghino diagonalmente, dovrete obbligatoriamente far sì che i pixel interni si colleghino ortogonalmente (vedere Figura 84b). Considerate il motivo di questo obbligo: se i pixel di cornice e quelli interni potessero essere collegati diagonalmente, i pixel interni potrebbero essere collegati ai pixel al di fuori della cornice in posizioni dove i pixel di cornice sono collegati diagonalmente.

Riempimenti semplici con linee orizzontali

Prima di affrontare le complessità degli algoritmi di riempimento aree, ricordate che potete riempire molte forme geometriche regolari senza dover utilizzare un particolare algoritmo. Un'applicazione diffusa di questa tecnica viene mostrata nel Listato 77. Questa routine riempie un'area rettangolare del buffer del video con pixel di un valore specifico. Essa risulta veloce in quanto la subroutine che traccia le linee orizzontali è veloce.



a.



b.

Figura 84. *Collegamento dei pixel. Nella Figura 84a, i pixel di cornice (neri) sono collegati ortogonalmente, mentre i pixel interni (grigi) sono collegati sia ortogonalmente sia diagonalmente. Nella Figura 84b, i pixel di cornice sono collegati sia ortogonalmente sia diagonalmente, in modo che i pixel interni siano collegati solo ortogonalmente.*

```
/* Listato 77 */

FilledRectangle( x1, y1, x2, y2, n )
int          x1,y1;                /* angolo superiore sinistro */
int          x2,y2;                /* angolo inferiore destro */
int          n;                    /* valore di pixel */
{
             int    y;

             for (y=y1; y<=y2; y++)       /* traccia rettangolo come
                                          /* una serie di linee
                                          /* orizzontali adiacenti*/
             Line( x1, y, x2, y, n )}:
```

Listato 77. *Riempimento di un rettangolo con linee orizzontali.*

La creazione di simili routine per tracciare triangoli, esagoni e cerchi pieni non è difficile, grazie alla regolarità e alla simmetria di queste forme. La scrittura di una routine generica che permetta di riempire poligoni convessi o irregolari risulta più complessa; in questo caso, dovete convertire a scansione ogni lato del poligono (usando, ad esempio, l'algoritmo di Bresenham del Capitolo 6) per creare un elenco dei pixel che definiscono il perimetro del poligono. Questo elenco contiene coppie di pixel che possono essere successivamente collegate con linee orizzontali per riempire l'interno del poligono.

CONSIGLIO

Diversi libri di buon livello trattano il problema della conversione a scansione e del riempimento di poligoni arbitrari. Consultate le librerie specializzate sull'argomento.

Sebbene le tecniche di riempimento di poligoni trovino molti impieghi, alcune applicazioni richiedono il riempimento di un'area che presenta confini completamente arbitrari, come ad esempio una cartina geografica o una forma irregolare che è stata tracciata interattivamente. In questo caso, la vostra routine di riempimento deve definire l'area utilizzando solo i valori di pixel del buffer del video. La parte restante di questo capitolo presenta gli algoritmi e i codici sorgente operativi per tre routine di questo tipo.

Tre algoritmi di riempimento aree

I tre algoritmi descritti qui sono stati tutti progettati soprattutto per i sistemi di visualizzazione IBM. Essi utilizzano le subroutine di manipolazione di pixel e di tracciamento linee sviluppate nei Capitoli 4, 5 e 6. Inoltre, tutti i tre algoritmi partono dal pressupposto che i pixel di cornice possono essere collegati diagonalmente e che i pixel interni devono essere collegati ortogonalmente (come nella Figura 84b). Potete quindi riempire aree con confini tracciati utilizzando le routine di tracciamento linee e di tracciamento ellissi dei Capitoli 6 e 7, dal momento che quelle routine tracciano figure collegate diagonalmente.

Per concludere, tutti i tre algoritmi possono riempire un'area che contiene un buco al suo interno (vedere Figura 85). Questo tipo di buchi è costituito da una serie di pixel di cornice non contigui ai pixel della cornice esterna dell'area. Ogni algoritmo è progettato per rilevare la presenza di buchi e di riempire correttamente i pixel interni che li circondano.

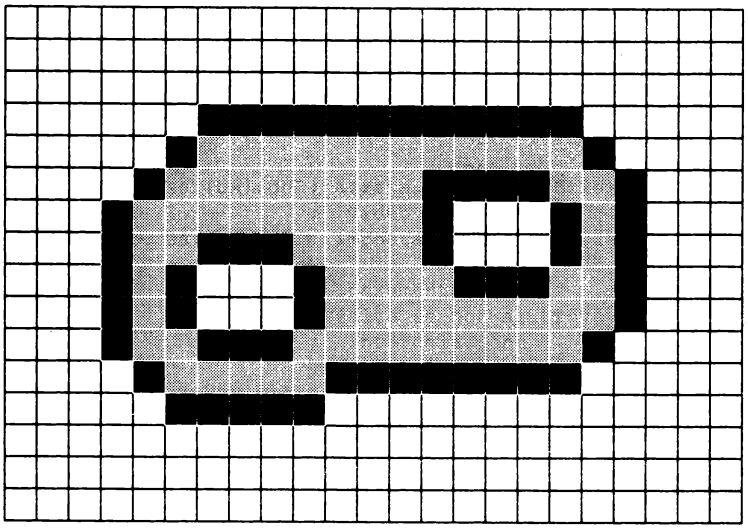


Figura 85. *Un'area il cui interno (pixel grigi) contiene due buchi*

Semplice riempimento ripetitivo

Un metodo per riempire un'area è quello di iniziare riempiendo un dato pixel "seme" al suo interno, e quindi di riempire ogni pixel adiacente al seme, quindi ogni pixel adiacente ai pixel adiacenti e così via fino a riempire l'intera area. La routine in C del Listato 78, *PixelFill()*, mostra questo procedimento. In *PixelFill()*, come negli altri algoritmi di questo capitolo, i pixel interni all'area vengono considerati collegati orizzontalmente e verticalmente, ma non diagonalmente (*PixelFill()* può essere facilmene modificata per riempire aree collegate diagonalmente, se necessario).

```
/* Listato 78 */

int             FillValue;          /* valore dei pixel nell'area riempita */
int             BorderValue;        /* valore dei pixel nella cornice */

PixelFill( x, y )
int             x,y;
{
                int     v;

                v = ReadPixel( x, y );

                if ( (v!=FillValue) && (v!=BorderValue) )
                {
                  SetPixel( x, y, FillValue );
```

```
            PixelFill( x-1, y );
            PixelFill( x+1, y );
            PixelFill( x, y-1 );
            PixelFill( x, y+1 );
          }
}
```

Listato 78. *Un semplice riempimento di aree ripetitivo.*

Prima di riempire un pixel, *PixelFill()* controlla il valore del pixel per determinare se è richiesto il riempimento. Se il pixel non è né un pixel di cornice né un pixel precedentemente riempito, la routine aggiorna il valore di pixel e si richiama ripetutamente. Dal momento che *PixelFill()* non riempie pixel riempiti in precedenza, la routine opera correttamente anche in quelle aree che presentano dei buchi.

Seppur semplice, *PixelFill()* risulta inefficiente. Una ragione è che in media solo una delle quattro chiamate ripetitive a *PixelFill()* produce un qualche risultato (ogni pixel può essere riempito solo una volta, ma ogni volta che un pixel viene riempito, vengono effettuate quattro chiamate ripetitive alla funzione. L'unica eccezione è rappresentata dal caso del pixel seme). Di conseguenza, *PixelFill()* non produce alcun risultato per circa il 75 percento del tempo, il che non può certo considerarsi efficiente.

CONSIGLIO

Un altro problema di *PixelFill()* è che il grado di ripetitività può essere aumentato oltre i limiti della memoria di stack disponibile. Ad esempio, lo spazio di stack di default per il codice generato dal compilatore C della Microsoft è di 2 KB. Potete facilmente superare questo limite utilizzando *PixelFill()* anche per riempire aree relativamente piccole.

Riempimento di linee adiacenti

Un approccio migliore è quello di considerare l'interno di un'area come se fosse un gruppo di segmenti di linee adiacenti che sono collegati verticalmente invece che come un gruppo di pixel collegati sia verticalmente sia orizzontalmente. Un algoritmo che riempie segmenti di linee adiacenti tende ad essere molto più efficiente di un riempimento ripetitivo pixel per pixel, in quanto verifica e riempie i pixel in modo più efficiente. Inoltre, questo concetto dell'area è molto più vicino alla rappresentazione fisica dei pixel nel buffer del video, all'interno del quale i pixel sono disposti in righe orizzontali da visualizzare durante la scansione del reticolo.

La routine del Listato 79, *LineAdjFill()*, implementa un algoritmo per linee adiacenti di riempimento di un'area. La sua strategia generale è quella di localizzare ogni gruppo di pixel collegati orizzontalmente all'interno dell'area. Come il semplice riempimento ripetitivo, anche questo algoritmo inizia da un pixel seme indicato come all'interno dell'a-

rea. La routine effettua la scansione verso destra e verso sinistra per trovare le estremità della riga del pixel seme, quindi riempie l'intera riga.

```
/* Listato 79 */

#define        UP                -1
#define        DOWN              1


LineAdjFill( SeedX, SeedY, D, PrevXL, PrevXR )
int            SeedX,SeedY;          /* seme per riga corrente di pixel */
int            D;                    /* direzione ricerca per scoprire
                                        riga corrente */
int            PrevXL,PrevXR;        /* estremità della riga precedente
                                        di pixel */
{
               int     x,y;
               int     xl,xr;
               int     v;

               y = SeedY;            /* inizializza a coordinate di seme */
               xl = SeedX;
               xr = SeedX;

               ScanLeft( &xl, &y );  /* determina estremità del segmento
                                        di linea seme */
               ScanRight( &xr, &y );

               Line( xl, y, xr, y,FillValue )
                                  ;/* riempie linea con FillValue */
/* trova e riempie segmenti di linea adiacenti nella stessa direzione */

               for (x=xl; x <= xr; x++)
                                  /* controlla righe adiacenti di pixel */
               {
                 v = ReadPixel( x, y+D );
                 if ( (v!=BorderValue) && (v!=FillValue) )
                   x = LineAdjFill( x, y+D, D, xl, xr );
               }

/* trova e riempie segmenti di linea adiacenti nella direzione opposta */

               for (x = xl; x< PrevXL; x++)
               {
                 v = ReadPixel( x, y-D );
                 if ( (v!=BorderValue) && (v!=FillValue) )
                   x = LineAdjFill( x, y-D, -D, xl, xr );
               }


               for (x=PrevXR; x < xr; x++
               {
                 v = ReadPixel( x, y-D );
```

```
                if ( (v!=BorderValue) && (v!=FillValue) )
                  x = LineAdjFill( x, y-D, -D, xl, xr );
              }

              return( xr );
}


ScanLeft( x, y )
int           *x,*y;

{
              int    v;


              do
              {
                --(*x);                    /* si sposta a sinistra di
                                              un pixel  */
                v = ReadPixel( *x, *y )    /* determina il suo valore */

              }
              while ( (v!=BorderValue) && (v!=FillValue) );

              ++(*x);                      /* coordinata x del pixel
                                                all'estrema sin. di riga */
}

ScanRight( x, y )
int           *x,*y;
{
              int    v;
              do
              {
                ++(*x);              /* si sposta a destra di un pixel */
                v = ReadPixel( *x, *y );  /* determina il suo valore */
              }
              while ( (v!=BorderValue) && (v!=FillValue) );

              --(*x);                /* coordin. x del pixel
                                         all'estrema destra di riga */
}
```

Listato 79. *Una routine di riempimento linee adiacenti.*

L'algoritmo procede localizzando tutti i gruppi di pixel collegati orizzontalmente che sono verticalmente adiacenti al gruppo appena passato. Ogni volta che ritrova un gruppo adiacente di pixel non ancora riempiti, *LineAdjFill()* viene chiamata ripetutamente per riempire i pixel. L'algoritmo termina quando tutti i pixel interni sono stati riempiti.

La Figura 86 mostra l'ordine in cui *LineAdjFill()* riempie una semplice area comprendente sette segmenti di linea. Si presuppone che il pixel seme si trovi all'interno del segmento di linea 1, e la routine viene inizialmente chiamata con una direzione di ricerca verso l'alto. La routine per prima cosa effettua la ricerca dei pixel non riempiti sulla riga

dei pixel al di sopra del seme (cioè, il segmento di linea 2). Dal momento che la riga non è stata ancora riempita, la routine viene richiamata ripetutamente per riempirla. Analogamente, i segmenti di linea 3 e 4 vengono riempiti tramite successive chiamate ripetitive a *LineAdjFill()*. A questo punto, né il segmento di linea 4 né il segmento di linea 3 contengono pixel non riempiti adiacenti ad essi, ma quando i pixel al di sotto del segmento di linea 2 vengono passati, il segmento di linea 5 viene rilevato e riempito. Infine, i segmenti di linea 6 e 7 vengono riempiti in successione.

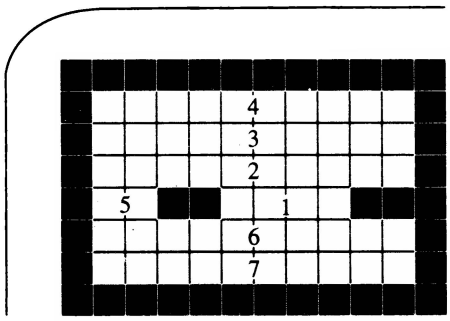


Figura 86. *Avendo un pixel seme nel segmento di linea 1, LineAdjFill() riempie i segmenti di linea adiacenti in questa area in ordine numerico.*

CONSIGLIO

Un grafico di linee adiacenti (LAG) è essenzialmente un diagramma dei collegamenti tra segmenti di linea adiacenti all'interno di un'area (vedere Figura 87). Il problema del riempimento di un'area è equivalente all'attraversamento del LAG relativo in modo tale che tutti i nodi all'interno del grafico siano stati toccati. In pratica, l'attraversamento del LAG è relativamente facile (esistono diversi libri sugli algoritmi sull'attraversamento grafico) in confronto alla generazione del grafico avendo solo i pixel del buffer del video (ciò che in effetti produce *LineAdjFill()*).

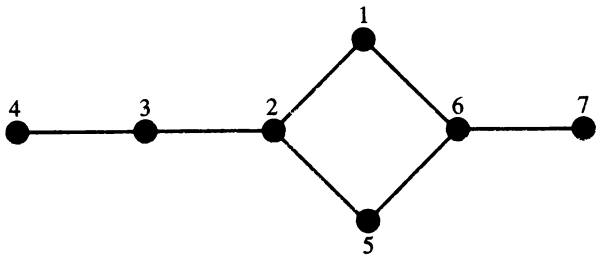


Figura 87. *Un semplice grafico di linee adiacenti (LAG).*

LineAdjFill() è molto più efficiente di *PixelFill()*, in quanto verifica raramente un pixel più di una volta per determinare se deve essere riempito. Ogni volta che viene richiamata la routine, essa riempie un segmento di linea, quindi verifica le righe adiacenti di pixel alla ricerca di pixel da riempire. La routine non esamina i pixel che sono già stati verificati nel corso della precedente chiamata (cioè, quei pixel tra *PrevXL* e *PrevXR*) e non esamina i pixel da riempire con successive chiamate (cioè, i pixel tra il valore corrente di x ed il valore ritornato da una chiamata a *LineAdjFill()*). La logica ripetitiva diventa chiara quando seguite l'esecuzione della routine mentre riempie un'area come quella schematizzata nella Figura 86.

CONSIGLIO

Se implementate un algoritmo di riempimento di linee adiacenti in linguaggio Assembler, potete migliorarne l'efficienza mantenendo uno stack di parametri ed eseguendo la funzione iterativamente invece che ripetutamente. L'ossatura dell'algoritmo diventa quindi

```
Ipush  ( ..  parametri iniziali su stack .. );
while  ( .. stack non vuoto .. )
  LineAdjFill();
```

La routine di riempimento preleva i parametri posti in alto sullo stack e inserisce nuove serie di parametri al posto di effettuare chiamate ricursive.

```
LineAdjFill
{
   pop  ( .. parametri correnti prelevati dallo stack .. )
   .
   .
   if  ( .. linea adiacente deve essere riempita .. )
```

```
        push  ( .. nuovi parametri .. )
}
```

Nel linguaggio Assembler, una singola istruzione in linguaggio macchina può eseguire ogni istruzione push e pop, migliorando enormemente le prestazioni dell'algoritmo.

Un algoritmo di linee adiacenti può essere adattato per riempire un'area con una maschera o pattern di pixel oltre che con un singolo valore di pixel. Per questo motivo, esso viene utilizzato diffusamente nei pacchetti di grafica commerciale (ne sono esempi BASICA dell'IBM e GW-BASIC della Microsoft). La modifica dell'algoritmo per eseguire riempimenti con un pattern comporta la sostituzione della routine di tracciamento linee orizzontali con una routine di tracciamento pattern e che il test che determina se un pixel è già stato riempito tenga conto dei valori di pixel del pattern di riempimento.

Queste modifiche possono sembrare innocue, ma possono ridurre drasticamente le prestazioni della routine di riempimento. La logica richiesta per rilevare la presenza di pixel precedentemente riempiti può essere complicata, in particolare se permettete che la configurazione di riempimento contenga pixel con lo stesso valore dei pixel di cornice.

Riempimento di cornice

Dal momento che la cornice di un'area definisce l'estensione dell'interno, è possibile riempire un'area seguendo i pixel collegati di cornice alle estremità dei segmenti di linee adiacenti che compongono la parte interna. Fintanto che riempite l'area contemporaneamente al tracciamento della cornice, però, questo tipo di algoritmo di riempimento che segue la cornice non offre alcun vantaggio preciso rispetto all'algoritmo di linee adiacenti.

Tuttavia, se separate il problema del seguire la cornice da quello del riempimento dell'interno dell'area, l'algoritmo che ne risulta diventa più flessibile. Il processo del riempimento di un'area si suddivide quindi in tre fasi separate:

1) Creazione di un elenco ordinato dei pixel di cornice (seguendo la cornice).

2) Scansione dell'interno dell'area alla ricerca di buchi.

3) "Collegamento dei punti" nell'elenco da sinistra verso destra con linee orizzontali quindi riempimento dell'area.

La routine *BorderFill()* del Listato 80 esegue il riempimento di un'area utilizzando questo metodo a tre fasi. L'algoritmo esegue iterativamente le tre fasi, una volta per i confini dell'area e una volta per ogni buco che si trova all'interno dell'area.

```
/*Listato 80*/

#define BLOKED        1
#define UNBLOKED      2
#define TRUE          1
#define FALSE         0

struct BP struct                /* tabella dei pixel di cornice*/

              int    x, y;
              int    flag;
)
              BP[3000];         /*(aumenta se necessario) */

int           BPstart;          /* inizio della tabella */
int           BPend =0;         /* prima cella vuota nella tabella */
int           FillValue         /* valore dei pixel nell'area riempita */
int           BorderValue       /* valore dei pixel nella cornice */


BorderFill( x, y )
int           x,y;
(
              do                     /* esegue finché intera tabella
                                        non è stata passata */
              (
              TracerBorder( x, y);   /* segue cornice a partire da x,y */
              SortBP( BP );          /* ordina la tabella di pixel
                                        di cornice */
              Scan Region( &x, &y ); /* ricerca buchi all'interno */
              )
              while (BPstart < BPend);

              FillRegion();                /* usa la tabella per riempire
                                              l'interno */
)

Scan Region( x, y )
int           *x, *y;
(
              int    i = BPstart;
              int    xr;

              while (i<BPend)
              (
                if (BP[i].flag == BLOCKED) /* salta pixel se di blocco */
                  ++i;

              else
              (                            /* se c'è almeno un pixel
                                              da riempire .. */
                 if(BP[i].x < BP[i+1].x-1) /*.. passa in scansione
                                              la linea */
                 (
                   xr = ScanRight( BP[i].x+1, BP[i].y );
                   if (xr<BP[i+1].x)       /* se viene trovato un pixel
                                              di cornice.. */
                   (
```

```
                    *x = xr;     /*.. ritorna le sue coordinate x,y */
                    *y = BP[i].y;
                  break;
                 )
               )

               i += 2;           /*  procede superando questa */
                                 /*    coppia di pixel */
             )
           }
           BPstart = i;
)
SortBP()                  */ usa routine di ordinamento rapido della
                             biblioteca Microsoft C */
(
           int    CompareBP();

           qsort( BP+BPstart, BPend-BPstart, sizeof(struct BPstruct,
            CompareBP );
)

CompareBP( arg1, arg2 )  /* ritorna  -1 se arg1 < arg2 */
struct  BPstruct  *arg1,*arg2;
)
           int    i;

           i = arg1->y - arg2->y;       /* ordina per coordinata y --
           if (i!=0)
             return( <i<0) ? -1 : 1 );  /* (ritorna -1 se i<0, 1 se i>0) */

           i = arg1->x - arg2->x;       /* ordina per coordinata x */
           if (i!=0)
              return( (i<0 ?  -1 : 1 );

           i = arg1->flag - arg2->flag; /* ordina per flag */
             return( (i<0) ? -1 : 1 );
)


FillRegion()
(
           int    i;

           for(i=0; i<BPend;)
           (
             if (BP[i].flag == BLOKED)  /* salta pixel se di blocco */
               ++i;

             else
             if (BP[i].y != BP[i+1].y)  /* salta pixel se ultimo */
                ++i;                    /*  della linea */

             else
             )                          /* se c'è almeno un pixel */
                                        /*  da riempire.. */
               if (BP[i].x < BP[i+].
```

```
                x  < BP[i+].x-1)          /* .. traccia una linea */
                Line( BP[i].xx+1, BP[i].y, BP[i+1].x-1, BP[i+1].y, FillValue)

               i +=2;
             )
            )
)

/* routine per seguire la cornice */

strct BPstruct CurrentPixel
int           D;                    /* direzione corrente di ricerca */
int           PrevD;                /* direzione precedente di ricerca*/
int           PrevV;                /* direzione precedente verticale */


TraceBorder( StartX, StartY )
int           StartX,StartY;
(
              int    NextFound;     /* flag */
              int    Done;

/* inizializza */

              CurrentPixel.x = StartX;
              CurrentPixel.y = StartY;

              D =6;                 /* direzione corrente di ricerca */
              PrevD = B;            /* direzione precedente di ricerca */
              Prev =2;              /* direzione più recente verticale */

/* loop intorno alla cornice fino a ritorno al pixel di partenza */
              da
              (
                NextFound = FindNextPixel();
                Done =
                 (CurrentPixel.x == StartX) && (CurrentPixel.y == StartY);
              )
              wile (NextFound && !Done;

/* se c'è un solo pixel nella cornice, aggiunge due volte alla tabella */
              if (!NextFound)       /* il pixel non ha pixel adiacenti */
              (
              AppendBPList( StartX, StartY, UNBLOCKED)

              AppendBPList( StartX, StartY, UNBLOCKED)
              )

/* se la direzione dell'ultima ricerca era verso l'alto, aggiunge il
pixel iniziale alla tabella */

              else
              if ( (PrevD < 3) && (PrevD> = 1) )
                AppendBPList( StartX, StartY, UNBLOCKED );
}
```

```
FindNextPixel( )
{
              int    i;
              int    flag;


              for (i=-1; i5;< i++)
              {
                flag =  FindBP((D+i)& 7);
                                      /* cerca prossimo pixel di cornice */
                if (flag)             /* flag è TRUE se trovato */
                {
                  D = (D+i) & 6;      /* (D+i) MOD 2 */
                  break;              /* esce da loop */
                }
              }

              return(flag);
}


FindBP( d )
int           d;                      /* direzione della ricerca del */
                                      /*   pixel successivo di cornice */
{
              int    x,y;

              x = CurrentPixel.x;
              y = CurrentPixel.y;


              NextXY( &x, &y, d );    /* rileva x,y del pixel nella */
                                      /*   direzione d */

              if ( BorderValue == ReadPixel( x, y ) )
              {
                AddBPList( d );       /* aggiunge pixel a x,y alla */
                                      /*   tabella */
                CurrentPixel.x=x;     /* pixel a x,y diventa pixel */
                                      /*   corrente */
                CurrentPixel.y = y;
                return( TRUE );
              }
              else
                return( FALSE );
}


NextXY( x, y, Direction )
int           *x,*y;
int           Direction;
{
              switch( Direction )          /*   3 2 1   */
              {                            /*   4   0   */
                case 1:                    /*   5 6 7   */
                case 2:
                case 3:
```

```
                    *y -= 1;        /* verso l'alto */
                    break;
                case 5:
                case 6:
                case 7:
                    *y += 1;        /* verso il basso */
                    break;
            }

            switch(Direction)
            {
                case 3:
                case 4:
                case 5:
                    *x -= 1;        /* verso sinistra */
                    break;
                case 1:
                case 0:
                case 7:
                    *x += 1;        /* verso destra */
                    break;
            }
}
AddBPList( d )
int         d;
{
            if (d == PrevD)
                SameDirection();

            else
            {
                DifferentDirection( d );
                PrevV = PrevD;              /* nuova direzione verticale
                                               precedente */
            }

            PrevD = d;              /* nuova direzione di ricerca */
                                    /*   precedente */
}

SameDirection()
{
            if (PrevD == 0)         /* spostamento verso destra... */
                BP[BPend-1].flag = BLOCKED;
                                    /* pixel precedente di blocco */

            else
            if (PrevD != 4)         /* se non è in corso spostamento */
                                    /*   orizzontale */
                AppendBPList( CurrentPixel.x, CurrentPixel.y, UNBLOCKED );
}

DifferentDirection( d )
int         d;
{
```

```
/* spostamento verso sinistra precedente */

              if (PrevD == 4)
              {
                if (PrevV == 5)       /* se dall'alto .. */
                  BP[BPend-1]
                  .flag = BLOCKED;    /* .. blocco all'estrema destra */
                                      /*   nella linea */

                AppendBPList( CurrentPixel.x, CurrentPixel.y,
                BLOCKED );
              }

/* spostamento precedente verso destra */

              else
              if (PrevD == 0)         /* spostamento precedente verso */
                                      /*   destra... */
              {
                BP[BPend-1].flag=BLOCKED; /* blocco all'estrema destra */
                                          /*  nella linea */

                if (d == 7)               /* se linea è iniziata */
                                          /*   dall'alto */
                  AppendBPList( CurrentPixel.x, CurrentPixel.y,BLOCKED );
                else
                  AppendBPList( CurrentPixel.x, CurrentPixel.y,UNBLOCKED );
              }
/* spostamento precedente in una direzione verticale */

              else
              {
                AppendBPList( CurrentPixel.x, CurrentPixel.y,UNBLOCKED );

/* aggiunge pixel due volte se minimo o massimo verticale locale */

              if ( ( (d>=1) && (d<3) ) && ( (PrevD>= 5) && (PrevD <=7) ) ||
                   ( (d>=5) && (d<7) ) && ( (PrevD>= 1) && (PrevD <=3) ) )
                AppendBPList( CurrentPixel.x, CurrentPixel.y,UNBLOCKED );
              }
}

AppendBPList ( p, q, f )
nt            p,q;                      /* coordinate x,y di pixel */
int           f;                        /* flag */
{
              BP[BPend].x = p;
              BP[BPend].y = q;
              BP[BPend].flag = f;

              ++BPend;                  /* incrementa oltre ultimo */
                                        /*   elemento della tabella */
}

/*routine per passare in scansione una linea per un pixel di cornice*/
int           Xmax;                     /* maggiore coordinata x di */
                                        /*  pixel valida */
```

```
ScanRight( x, y )
int             x,y;
{
                while ( ReadPixel( x, y ) != BorderValue )
                {
                  ++x;                  /* incrementa x */
                  if (x==Xmax)          /* se fine linea nel buffer... */
                    break;              /* .. esce dal loop */
                }
                return( x );
}
```

Listato 80. *Una routine di riempimento aree che segue la cornice dell'area.*

Il modulo *TraceBorder()* crea una tabella che contiene l'indirizzo di ogni pixel della cornice dell'area. *SortBP()* ordina la tabella dei pixel di cornice incrementando le coordinate *y* e *x*. La routine *ScanRegion()* esamina il segmento di linea interno che si trova tra ogni coppia di pixel di cornice nella tabella. Se rileva un pixel di cornice all'interno del segmento di linea, *ScanRegion()* presume di aver incontrato un buco all'interno dell'area; la routine ritorna quindi le coordinate (*x,y*) del pixel di cornice in modo che *TraceBorder()* e *SortBP()* possano aggiornare la tabella con i pixel di cornice del buco. Questo processo continua fino a che tutta l'area interna è stata esaminata. Successivamente *FillRegion()* usa l'elenco ordinato di pixel di cornice per riempire l'area tracciando una linea orizzontale tra ogni coppia di pixel della lista.

TraceBorder() inizia con un pixel seme sulla cornice di destra dell'area. Procede in senso orario da pixel a pixel sulla cornice. Dal momento che la ricerca procede in senso orario, l'interno dell'area è sempre a destra della direzione della ricerca. Se un pixel non è adiacente all'interno, l'algoritmo non lo identifica come pixel di cornice. L'algoritmo si assicura che i pixel di cornice da esso rilevati siano veramente adiacenti all'interno esaminando sempre prima i pixel alla destra della direzione di ricerca.

L'algoritmo identifica la propria direzione di ricerca con uno degli otto codici numerici mostrati nella Figura 88. Di conseguenza, nella Figura 89, l'algoritmo si sposta dal pixel *b* al pixel *c* nella direzione 6 (discendente). Per trovare il pixel successivo sulla cornice, l'algoritmo inizia dall'esame del pixel alla destra della direzione 6, cioè nella direzione 4. Questo pixel non è un pixel di cornice, ma lo è il pixel nella direzione 5 (pixel *d*), quindi *d* viene aggiunto all'elenco. L'algoritmo continua a seguire la cornice fino a ritornare al pixel di partenza (la ricerca si interrompe immediatamente nel caso di una "cornice" degenere costituita da un solo pixel).

TraceBorder() esegue un altro compito oltre all'identificazione dei pixel di cornice. Essa indica anche se ogni pixel di cornice definisce l'estremità sinistra o destra di un segmento di linea orizzontale interno (dal momento che *FillRegion()* traccia linee orizzontali da sinistra verso destra, *TraceBorder()* contrassegna ogni pixel di cornice con un flag indicante se il pixel può essere utilizzato come cornice sinistra). Inoltre, se un pixel può essere utilizzato come cornice sia destra sia sinistra (vedere Figura 90), *TraceBorder()* lo aggiunge due volte alla tabella. La logica di *SameDirection()* e di *DifferentDirection()* realizza questi compiti.

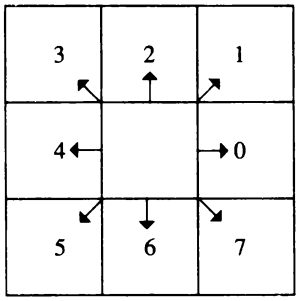
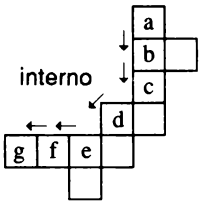


Figura 88. *Codici numerici per le direzioni di percorso dei pixel di cornice.*



	Direzione
a → b	6
b → c	6
c → d	5
d → e	5
e → f	4
f → g	4

Figura 89. *Identificazione di pixel di cornice in TraceBorder().*

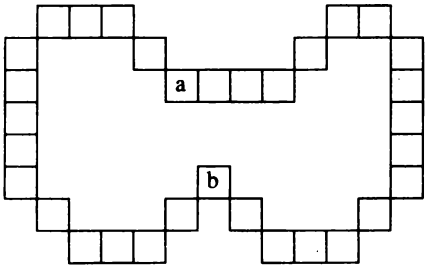


Figura 90. *I pixel possono delimitare l'interno in direzione sinistra, destra o in entrambe le direzioni: il pixel* a *è un pixel di cornice sulla destra di una riga di pixel interni; esso è bloccato sulla destra da altri pixel di cornice. Il pixel* b *è sia un pixel di cornice sinistra sia un pixel di cornice destra.*

TraceBorder() potrebbe sembrare complessa, tuttavia è una routine relativamente veloce. Le fasi più lente di *BorderFill()* sono in realtà *SortBP()*, che ordina la tabella di pixel di cornice, e *ScanRegion()*, che ricerca i pixel di cornice all'interno dell'area. Se

SortBP() e *ScanRegion() sono lente, anche BorderFill()* sarà lenta, in quanto queste routine vengono eseguite iterativamente, una per ogni buco dell'area.

Potete migliorare significativamente le prestazioni di *BorderFill()* modificando *TraceBorder()* in modo che crei un elenco di pixel di cornice nel corretto ordine come prima cosa, evitando completamente l'operazione di ordinamento. Potete creare in modo efficiente l'elenco ordinato utilizzando una qualsiasi strutturaU di dati, compreso un elenco correlato o una tabella a dimensione fissa. Questo tipo di modifica è particolarmente efficace quando l'algoritmo viene usato per riempire aree che contengono uno o più buchi. Al posto di ordinare l'elenco ogni volta che viene rilevato un buco, l'algoritmo modificato inserisce semplicemente nell'elenco i pixel di cornice del buco.

La realizzazione di *ScanRegion()* in un linguaggio ad alto livello è relativamente facile, ma dal momento che la routine esamina tutti i pixel dell'interno dell'area, dovreste scriverla in linguaggio Assembler in modo che venga eseguita rapidamente. Inoltre, l'uso del linguaggio Assembler su EGA, VGA e su scheda InColor offre un preciso vantaggio, in quanto l'hardware di controllo grafico di questi sistemi può esaminare otto pixel contemporaneamente ed indicare quale, eventualmente, corrisponde al valore di pixel di cornice. La routine in lingaggio Assembler *ScanRight()* del Listato 81, che può essere usata nelle modalità grafiche a 16 colori EGA e VGA, opera ad una velocità 50 volte superiore rispetto alla versione in C del Listato 80.

```
                TITLE   'Listato 81'
                NAME    ScanRight10
                PAGE    55,132

;
; Nome:         ScanRight10
;
; Funzione:     Ricerca un pixel di un dato valore nelle modalità grafiche a
;               16 colori EGA/VGA
;
; Chiamante:    Microsoft C:
;
;                       int ScanRight10(x,y);
;
;                       int x,y;                    /* pixel iniziale */
;
;                       extern   int BorderValue;   /* valore pixel di cornice */
;
;               Ritorna la coordinata x del pixel di cornice all'estrema destra.
;

ARGx            EQU     word ptr [bp+4] ; indirizzamento di cornice di stack
ARGy            EQU     word ptr [bp+6]

ByteOffsetShift         EQU         3   ; usato per convertire i pixel in
                                        ;  offset di byte
BytesPerLine    EQU     80              ; 80 per maggior parte delle modal.
                                        ;  grafiche a 16 colori
                                        ; (40 per modalità 16 colori 320x200)
```

```
DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr10:near

                PUBLIC _ScanRight10
_ScanRight10    PROC    near

                push    bp              ; mantiene registri del chiamante
                mov     bp,sp
                push    si
                push    di

; calcola indirizzo di pixel di (0,y)

                mov     ax,ARGy         ; AX := y
                xor     bx,bx           ; BX := 0
                call    PixelAddr10     ; ES:BX -> buffer
                mov     di,bx           ; ES:DI -> buffer

; calcola offset di x nella riga

                mov     ax,ARGx
                mov     si,ax           ; SI,AX := x
                mov     cl,ByteOffsetShift
                shr     si,cl           ; SI := offset di x nella riga y
                add     di,si           ; DI := offset di x nel buffer
; calcola una maschera di bit per il primo byte da ricercare

                mov     cl,al
                and     cl,7            ; CL := x & 7
                mov     ch,0FFh
                shr     ch,cl           ; CH := maschera di bit per il primo
                                        ;  byte ricercato

; configura il controller grafico

                mov     dx,3CEh         ; DX := indirizzo di porta del
                                        ;  controller grafico

                mov     ah,_BorderValue ; AH := valore di pixel per reg.
                                        ;  confronto colore
                mov     al,2            ; AL := numero reg confronto colore
                out     dx,ax

                mov     ax,805h         ; AH := 00001000b (modalità di
                                        ;  lettura 1)
                out     dx,ax           ; AL := numero reg modalità

                mov     ax,0F07h        ; AH := 00001111b (valore reg
                                        ;  confronto colore)
                out     dx,ax           ; AL := numero reg confronto colore
```

```
; esamina primo byte per rilevare pixel di cornice

            mov     al,es:[di]      ; AL := bit non zero corrispondenti a
                                    ;  pixel di cornice
            inc     di              ; ES:DI - byte succ. da ricercare
            and     al,ch           ; applica maschera di bit
            jnz     L01             ; salta se trova pixel di cornice

; scorre resto della linea alla ricerca di pixel di cornice

            mov     cx,BytesPerLine
            sub     cx,si           ; CX := BytesPerLine - (offset di
                                    ;  byte di
                                    ;  pixel iniziale)
            dec     cx              ; CX := # di byte da scorrere

            repe    scasb           ; ricerca fino a lettura di byte non
                                    ;  0; cioè se trova pixel di cornice

; calcola valore x del pixel di cornice

            mov     al,es:[di-1]    ; AL := ultimo byte confrontato

L01:        sub     di,bx           ; DI := offset del byte oltre a
                                    ;  quello che contiene
                                    ;  un pixel di cornice
            mov     cl,ByteOffsetShift
            shl     di,cl           ; DI := coordinata x del primo pixel
                                    ;  nel byte

            mov     cx,8            ; CX := limite di loop

L02:        shl     al,1            ; isola primo pixel di cornice
            jc      L03

            loop    L02

L03:        sub     di,cx           ; DI := coordinata x del pixel di
                                    ;  cornice

; ripristina stato di default del controller grafico e ritorna al
; chiamante

            mov     ax,2            ; AH := 0 (valore di default di
                                    ;  confronto colore)
            out     dx,ax           ; ripristina reg confronto colore

            mov     al,5            ; AH := 0, AL := 5
            out     dx,ax           ; ripristina reg modalità

            mov     ax,di           ; AX := ritorna valore

            pop     di              ; ripristina reg. chiamante e ritorna
            pop     si
            mov     sp,bp
            pop     bp
            ret
```

```
_ScanRight10   ENDP

_TEXT          ENDS

_DATA          SEGMENTword public 'DATA'

               EXTRN  _BorderValue:byte

_DATA          ENDS

               END
```

Listato 81. *Una versione in linguaggio Assembler di ScanRight().*

La fase più veloce di *BorderFill()* è rappresentata dal riempimento stesso, in quanto le linee orizzontali possono essere tracciate velocemente. Di conseguenza, se dovete riempire ripetutamente la stessa area o copiare diverse volte la stessa area riempita, potete mantenere l'elenco di pixel di cornice creato la prima volta che si è eseguita *BorderFill()*. Questo accelera enormemente i riempimenti successivi, in quanto potete saltare le fasi di percorso cornice e di ordinamento.

Confronto degli algoritmi

Quale algoritmo di riempimento aree è il migliore? Ogni algoritmo descritto in questo capitolo ha pro e contro. Potete confrontarli in diversi modi. Un confronto valido deve tenere in considerazione il grado di semplicità dell'algoritmo, la velocità del codice compilato e l'adattabilità di ogni algoritmo a casi particolari di riempimento di aree.

L'algoritmo ripetitivo, pixel per pixel, implementato come *PixelFill()* è quanto di più semplice si possa avere. Il codice sorgente è breve e facile da implementare in linguaggio Assembler oltre che in un linguaggio ad alto livello. Tuttavia, *PixelFill()* è troppo inefficiente e troppo ripetitivo per essere utile in ogni occasione.

L'algoritmo di riempimento linee adiacenti *LineAdjFill()* è più complicato di *PixelFill()*. Ciononostante, *LineAdjFill()* migliora le prestazioni di *PixelFill()* in quanto esamina gruppi di pixel al posto di pixel individuali. *LineAdjFill()* inoltre opera più velocemente quando viene scritto per accedere al buffer del video con incrementi di un byte al posto di incrementi di un pixel. *LineAdjFill()* risulta anche molto meno ripetitivo di *PixelFill()*, quindi le sue richieste di memoria operativa sono ridotte rispetto a quelle di *PixelFill()*.

L'algoritmo a tre fasi implementato in *BorderFill()* è più complicato e in qualche modo più lento degli altri due algoritmi. Il vantaggio derivante dall'uso di *BorderFill()* risiede nella sua universalità. I suoi moduli possono essere prontamente adattati per alternare

i tipi di riempimento aree, comprendendo anche riempimenti con pattern e riempimenti di aree definite sotto forma di elenco numerico di coordinate (*x,y*).

Le prestazioni di *BorderFill()* dipendono dal numero di buchi dell'area. Essa risulta più veloce di *LineAdjFill()* nel riempimento di un'area senza buchi. Tuttavia, quando l'area da riempire ha l'aspetto di un colabrodo, *BorderFill()* rallenta moltissimo, in quanto deve aggiornare l'elenco ordinato di pixel di cornice ogni volta che esegue il riempimento del contorno di un buco.

Ciononostante, *BorderFill()* può fare diverse cose non realizzabili con gli altri algoritmi. Ad esempio, può riempire in modo affidabile aree che contengono pixel precedentemente riempiti, infatti a differenza di *BorderFill()*, sia *PixelFill()* sia *LineAdjFill()* si basano sul pressupposto implicito che nessun pixel interno ha lo stesso valore del valore di riempimento. Di conseguenza, *BorderFill()* riempie correttamente l'area mostrata nella Figura 91, mentre entrambe le altre routine falliscono lo scopo.

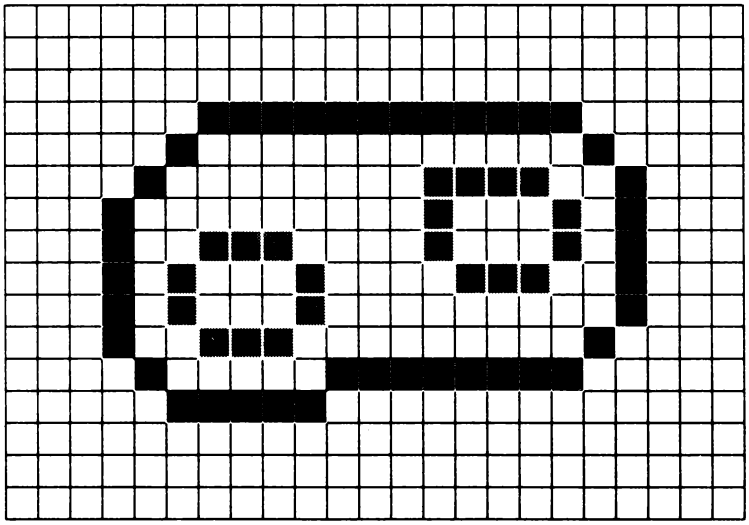


Figura 91. *Un caso di verifica degli algoritmi di riempimento. Né PixelFill() né LineAdjFill() possono riempire correttamente quest'area con pixel grigi, in quanto i "buchi" vengono trattati come se fossero già stati riempiti.*

CONSIGLIO

Potete modificare una routine come *LineAdjFill()* in modo che il suo rilevamento dei buchi all'interno dell'area non dipenda dalla presenza di pixel precedentemente riempiti. Ciò significa che l'algoritmo deve in qualche modo tenere conto dei pixel che ha già riempito. Un modo per fare ciò è quello di tenere conto dei punti dove la cornice incontra un minimo o massimo locale (vedere Figura 92). Queste locazioni possono identificare la parte superiore e la parte inferiore di un buco dell'area, permettendo all'algoritmo di riempimento di determinare quando interrompere il percorso intorno al buco.

Per alcune applicazioni *BorderFill()* presenta un forte vantaggio sugli altri algoritmi, in quanto le sue fasi di tracciamento cornice e di ordinamento dati generano un elenco di coordinate di pixel in forma numerica. L'elenco definisce in modo completo un'area bidimensionale di pixel. Potete tradurre o modificare la scala dell'area applicando le conversioni appropriate all'elenco di pixel di cornice. Fintanto che mantenete l'ordine dei pixel all'interno dell'elenco, potete utilizzare la routine *FillRegion()* in *BorderFill()* per riempire l'area definita dall'elenco. Per questo motivo, l'algoritmo *BorderFill()* è più adatto alle applicazioni che devono copiare aree arbitrarie, modificare la loro scala o dimensione o tracciarle ripetutamente nel buffer del video.

Inoltre, modificando la routine di linea orizzontale di *BorderFill()* potete facilmente riempire un'area con un pattern arbitrario o permettere le operazioni di pixel AND, OR e XOR. Sebbene possiate potenziare in questo modo *PixelFill()* o *LineAdjFill()*, il codice sorgente può diventare complesso in quanto questi algoritmi devono esaminare i pixel per determinare se sono stati già riempiti o meno.

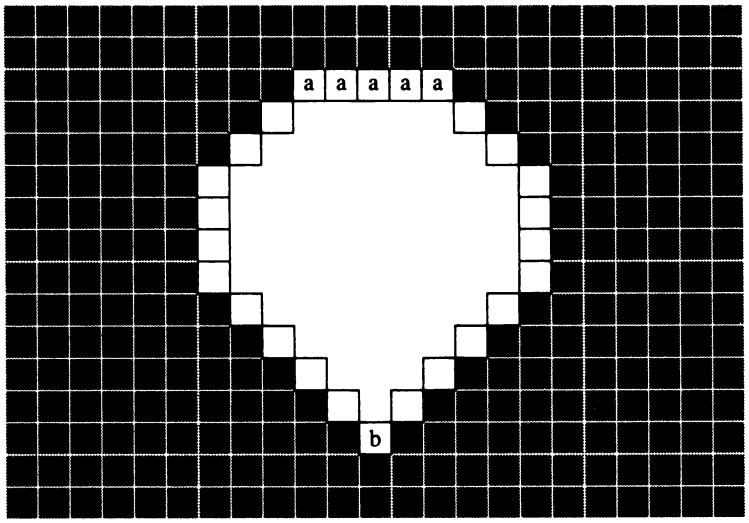


Figura 92. *Un algoritmo può rilevare la presenza di un buco all'interno di un'area localizzando il minimo e il massimo locale della cornice. I pixel contrassegnati con* a *indicano un massimo locale. I Pixel contrassegnati con* b *indicano un minimo locale.*

I compromessi tra complessità e prestazioni in questi agoritmi lasciano molto al vostro giudizio di programmazione. Nessun algoritmo singolo di riempimento aree rappresenta l'ideale per ogni possibile applicazione grafica. La vostra scelta di implementazione dovrebbe dipendere dalle prestazioni richieste, dai requisiti dell'applicazione stessa, dalle capacità del vostro hardware di visualizzazione e dallo sforzo che potete permettervi nell'integrare e nell'ottimizzare il codice.

9
Testo grafico

Tabelle di definizione caratteri
Supporto del BIOS del video
Creazione di una tabella di definizione caratteri

Generatori di caratteri software
Supporto del BIOS del video - Gestione dei pixel

Progettazione di un generatore di caratteri software
Allineamento orizzontale
Dimensioni variabili dei caratteri - Definizione
Orientamento dei caratteri
Collaborazione con il BIOS del video
Più potenza, più complessità

Implementazione di un generatore di caratteri software
CGA - HGC e HGC+ - MCGA
EGA e VGA - Scheda InColor

Pochi programmi sono corredati di un qualche tipo di visualizzazione di testo. La maggior parte delle applicazioni grafiche comprendono il testo sotto forma di immagini grafiche. Nelle modalità grafiche, il software che traccia i caratteri richiede la stessa attenta progettazione e costruzione richiesta dalle routine che tracciano figure geometriche come linee ed ellissi.

Nelle modalità alfanumeriche, naturalmente, la visualizzazione del testo è semplice. Introducete semplicemente il codice e l'attributo di un carattere nel buffer del video e lasciate al generatore di caratteri hardware il compito di posizionare i pixel sullo schermo. Nelle modalità grafiche, invece, il vostro programma deve memorizzare ogni pixel di ogni carattere nel buffer del video.

Questo capitolo spiega come tradurre i codici di carattere in configurazioni di pixel che compongono i caratteri nelle modalità grafiche. Gli esempi di programmazione sono specifici per l'hardware indicato, naturalmente, ma potete adattare il generatore di caratteri guidato da tabella descritto nel capitolo per utilizzare gli esempi con altri computer ed altre applicazioni grafiche.

Tabelle di definizione caratteri

Ogni carattere prodotto da un sistema di visualizzazione IBM è costituito da un insieme di pixel contigui. I pixel sono disposti in modo da apparire sullo schermo come caratteri riconoscibili e chiari. L'insieme di pixel che rappresenta un carattere è identico a prescindere dalla posizione del carattere sullo schermo o all'interno del buffer.

Il metodo più comodo per descrivere gli insiemi di pixel che rappresentano i caratteri di un set di caratteri è quello di creare una tabella all'interno della quale le configurazioni di bit rappresentano gli insiemi di pixel. Una tabella di definizione caratteri di questo tipo contiene una configurazione di bit per ogni carattere visualizzabile (vedere Figura 93). La configurazione di bit di ogni carattere viene definita all'interno di una matrice rettangolare. Quando la matrice di carattere ha la stessa dimensione per tutti i caratteri della tabella, e le definizioni della tabella sono disposte in ordine di codice di carattere, la conversione di un codice di carattere in un offset nella tabella risulta un'operazione semplicissima.

Potete utilizzare una tabella di definizione caratteri formattata in questo modo sia nelle modalità alfanumeriche sia in quelle grafiche nei sistemi di visualizzazione che supportano definizioni di caratteri alfanumerici basate su RAM. Il Capitolo 10 tratta in dettaglio questo argomento.

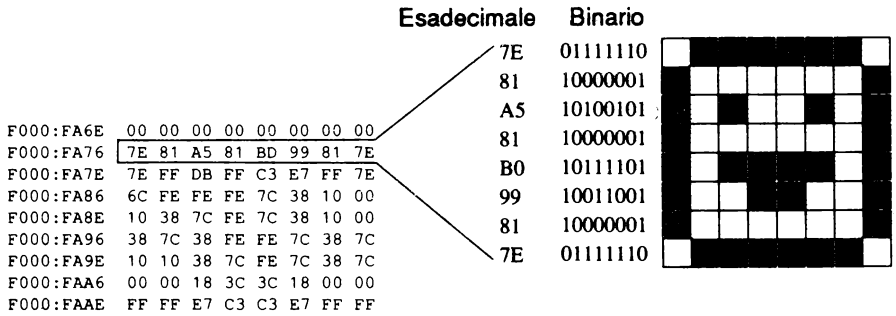


Figura 93. *L'inizio delle configurazioni di bit che indicano le definizioni di carattere 8 per 8 del BIOS ROM dell'IBM.*

Supporto del BIOS video

Il BIOS ROM dei PC e dei PS/2 contiene le tabelle di definizione caratteri di default da utilizzarsi nelle modalità grafiche. La dimensione dei caratteri nella tabella dipende dalla risoluzione verticale della modalità video. Nelle modalità video compatibili CGA a 200 linee, la matrice di carattere di default è larga 8 pixel e alta 8 pixel; nelle modalità grafiche a 350 linee, è larga 8 e alta 14; nelle modalità a 400 e a 480 linee è 8 per 16. In tutte le modalità grafiche, i caratteri di default sono larghi 8 pixel semplicemente perché ci sono 8 bit in un byte. Dal momento che ogni byte in una tabella di definizione caratteri rappresenta 8 pixel orizzontali, la definizione di caratteri come multipli di 8 pixel di larghezza rende semplice la manipolazione software della tabella.

Non viene applicato alcun limite equivalente all'altezza dei caratteri definiti in una tabella di definizione caratteri. In pratica, però, la matrice di carattere utilizzata sui sistemi di visualizzazione IBM dovrebbe raramente essere più piccola di 8 per 6 pixel o più grande di 8 per 16 pixel. Con una matrice di carattere al di fuori di questa gamma, l'altezza e la larghezza visualizzate dei caratteri diventano sproporzionate e i caratteri tendono ad apparire troppo corti o troppo allungati per essere letti facilmente.

Caratteri CGA di default

La Figura 93 mostra l'inizio della tabella di definizione caratteri per il set di caratteri di default delle modalità grafiche CGA. La tabella contiene una definizione a 8 byte per ognuno dei primi 128 caratteri ASCII (da 0 a 7FH). I primi otto byte della tabella corrispondono al codice di carattere 0, i secondi otto byte al codice di carattere 1 e così via. La configurazione di bit di ogni gruppo di otto byte rappresenta l'insieme di pixel visualizzato per la riga corrispondente di pixel nel carattere. Il primo degli otto byte di ogni gruppo corrisponde alla riga superiore di otto pixel.

Questa tabella di definizioni di caratteri 8 per 8 si trova a F000:FA6E nella ROM della piastra madre di tutti i PC e i PS/2. Tuttavia, la tabella definisce solo i primi 128 carat-

teri ASCII. Le definizioni dei caratteri del secondo gruppo di 128 codici ASCII (da 80H a 0FFH) si trovano in una tabella il cui indirizzo è memorizzato nel vettore di interrupt 1FH (0000:007C). Dal momento che il BIOS della piastra madre non contiene alcuna definizione per questi caratteri, l'indirizzo viene inizializzato a 0000:0000. Se utilizzate il BIOS ROM per visualizzare i caratteri ASCII da 80H a 0FFH nelle modalità grafiche CGA senza puntare questo vettore di interrupt su una tabella di definizione caratteri, i "caratteri" che vedrete sullo schermo corrisponderanno a qualsiasi configurazione binaria che si trovi nei primi 1024 byte di RAM.

CONSIGLIO

Il programma di utilità GRAFTABL dell'MS-DOS memorizza una tabella di definizione dei caratteri da 80H a 0FFH nella RAM ed aggiorna il vettore di interrupt 1FH in modo che la punti. I caratteri definiti in GRAFTABL sono identici a quelli visualizzati dal generatore di caratteri alfanumerici per i codici ASCII da 80H a 0FFH.

Caratteri di default EGA, VGA e MCGA

Il BIOS ROM dei sottosistemi di EGA, VGA e MCGA contiene definizioni per tutti i 256 codici ASCII per tutte le modalità grafiche (potete accedere direttamente a queste tabelle; i loro indirizzi possono essere ricavati richiamando la funzione 11H di INT 10H con AL = 30H). Quando selezionate una modalità grafica con la funzione 0 di INT 10H, il BIOS del video carica l'indirizzo della tabella di definizione caratteri appropriata per la modalità grafica nel vettore di interrupt 43H (0000:010C). Nelle modalità grafiche a 200 linee compatibili CGA, il BIOS punta anche il vettore di interrupt 1FH nella zona in cui vengono definiti i caratteri da 80H a 0FFH.

Creazione di una tabella di definizione caratteri

Il metodo più semplice per realizzare una tabella di definizione di caratteri è quello di utilizzare una delle tabelle di default del BIOS. Se i caratteri normali e classici di quelle tabelle non sono di vostro gradimento, potete trovare molti altri set di caratteri in commercio, oppure nell'ambito del software di pubblico dominio.

CONSIGLIO

Diversi set di caratteri standard sono definiti e depositati presso l'International Standards Organization (ISO). L'IBM fa riferimento a questi set di caratteri come a *pagine di codice* ed ha assegnato ad essi numeri identificativi arbitrari. Ad esempio, il set di caratteri ASCII standard del PC IBM è identificato dal numero di pagina di codice 437; la pagina di codice franco-canadese è la 863; la pagina di codice 850 corrisponde al set di caratteri generico "multi-lingue" studiato dall'IBM per le lingue che uti-

lizzano un alfabeto latino. Sia l'MS-DOS (a partire dalla versione 3.3) sia l'OS/2 permettono alle applicazioni di passare da una pagina di codice all'altra utilizzando un EGA e o un VGA. Quando un programma visualizza i caratteri tramite chiamate di funzione di sistema operativo, il sistema operativo utilizza le definizioni di caratteri della pagina di codice correntemente selezionata. Le applicazioni che impiegano set di caratteri di lingue diverse dall'inglese dovrebbero, se possibile, sfruttare le pagine di codice supportate dal sistema operativo.

Quando definite un vostro set di caratteri, potete selezionare uno dei diversi metodi alternativi. L'alternativa peggiore è quella di creare una propria tabella di definizione caratteri specificando ogni byte in codice sorgente. La Figura 94 mostra l'inizio di una tabella di questo tipo. Un'alternativa più elegante è quella di utilizzare un editor di set di caratteri. Con questi tipi di editor, si utilizzano i tasti di controllo cursore o un dispositivo di puntamento come una penna ottica o un mouse per specificare le configurazioni di bit nella tabella. Gli editor di set di caratteri sono anche disponibili sul mercato (potete anche scriverne uno vostro, utilizzando le routine contenute in questo libro).

Un altro approccio è quello di iniziare con uno dei set di caratteri standard del BIOS, modificandone le configurazioni di bit in un modo opportuno. Ad esempio, potreste invertire le configurazioni di bit di una tabella convertendo gli 0 in 1 e gli 1 in 0 (cioè, applicando un NOT logico ad ogni byte della tabella), creando così un set di caratteri "invertito".

```
CharDefs        db      000h,000h,000h,000h,000h,000h,000h,000h ; character 0
                db      03Ch,066h,0C0h,0C0h,0C0h,066h,03Ch,000h ; character 1
                db      0FCh,066h,066h,07Ch,06Ch,066h,0E6h,000h ; character 2
                db      0FEh,062h,068h,078h,068h,062h,0FEh,000h ; character 3
                db      078h,0CCh,0CCh,078h,0CCh,0CCh,078h,000h ; character 4
                db      078h,030h,030h,030h,030h,030h,078h,000h ; character 5
                db      0CCh,0CCh,0CCh,0CCh,0CCh,078h,030h,000h ; character 6
                db      0FEh,062h,068h,078h,068h,062h,0FEh,000h ; character 7
```

Figura 94. *Una tabella di definizione caratteri codificata a mano.*

Generatori di caratteri software

Una routine software che usa le configurazioni di bit di una tabella di definizione caratteri per tracciare i caratteri nel buffer del video viene chiamata generatore di caratteri software. Un generatore di caratteri software esegue diverse funzioni. Esso localizza la configurazione di bit per un dato codice di carattere, traduce la configurazione di bit nella corrispondente configurazione di pixel ed aggiorna i pixel ad una locazione specificata nel buffer del video.

Supporto del BIOS video

Il BIOS del video fornisce un generatore di caratteri software che viene usato ogni volta che vengono richiamate le funzioni 09H, 0AH, 0EH e 13H di INT 10H nelle modalità grafiche. Il generatore di caratteri software del PC e dell'AT IBM utilizza solo i caratteri 8 per 8 definiti a F000:FA6E e all'indirizzo indicato dal vettore di interrupt 1FH. La versione del BIOS dell'EGA e del PS/2 utilizza la tabella puntata dal vettore di interrupt 43H; questa versione determina l'altezza dei caratteri visualizzati a partire dalla variabile POINTS del BIOS che si trova a 0040:0085.

Potete impiegare il generatore di caratteri software per visualizzare i caratteri a partire da una qualsiasi tabella di definizione caratteri aggiornando gli appropriati vettori di interrupt con gli indirizzi della tabella. Su EGA e su PS/2, usate la funzione 11H di INT 10H per eseguire questa operazione.

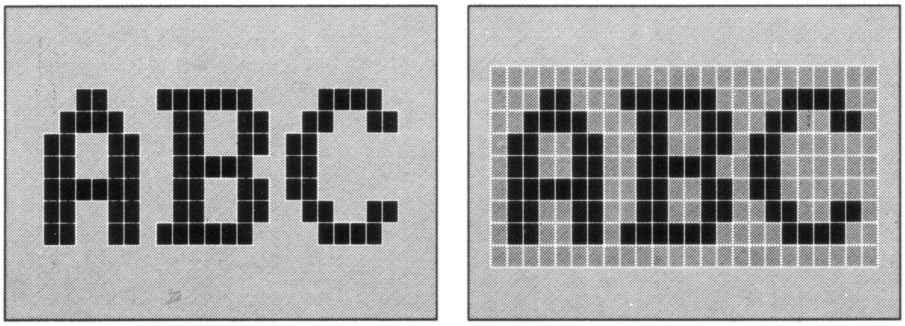
Il generatore di caratteri del BIOS è comodo, ma è in qualche modo limitato nelle capacità. In particolare, esso può memorizzare solo caratteri allineati per byte nel buffer del video. Se siete disposti a sacrificare la compatibilità con l'interfaccia INT 10H, potete scrivere un generatore di caratteri software più veloce che risulti più potente della versione standard del BIOS video.

Gestione dei pixel

I caratteri vengono memorizzati nel buffer del video modificando i valori dei gruppi di pixel appropriati. Potete aggiornare il buffer del video semplicemente sostituendo i vecchi valori di pixel con quelli nuovi. Per aggiornare i pixel potete anche eseguire delle operazioni logiche binarie (AND, OR o XOR).

La routine per visualizzare il testo nelle modalità grafiche può gestire i pixel di sfondo della matrice di carattere in due modi. Uno è quello di mantenere il più possibile intatto il contenuto del buffer del video aggiornando solo i pixel di primo piano, cioè aggiornando solo quei pixel che rappresentano il carattere in sé (vedere Figura 95a). L'altro metodo è quello di aggiornare tutti i pixel di primo piano e di sfondo all'interno dei confini della matrice di carattere rettangolare (vedere Figura 95b).

L'aggiornamento dei soli pixel di primo piano del carattere mantiene il numero maggiore possibile di pixel nel buffer del video. Questo potrebbe essere il metodo migliore per visualizzare del testo sovrapposto ad un'immagine grafica dettagliata o formata da un modello grafico. Tuttavia, la lettura dei caratteri visualizzati potrebbe risultare difficoltosa se l'immagine grafica in qualche modo si fonde con i caratteri. Ad esempio, il testo risulta invisibile all'interno di un'area riempita con pixel che hanno lo stesso valore dei pixel di primo piano del carattere.



a. b.

Figura 95. *Caratteri scritti senza pixel di sfondo (a.) e con pixel di sfondo (b.).*

Per evitare questo tipo di problemi, potete aggiornare tutti i pixel di primo piano e di sfondo nella matrice di carattere ogni volta che memorizzate un carattere nel buffer. Ciò evita la mascheratura indesiderata dei caratteri da parte di un modello grafico di sfondo

Lo svantaggio è che ogni volta che memorizzate un carattere nel buffer dovete sostituire il contenuto precedente del buffer con un carattere di riempimento rettangolare.

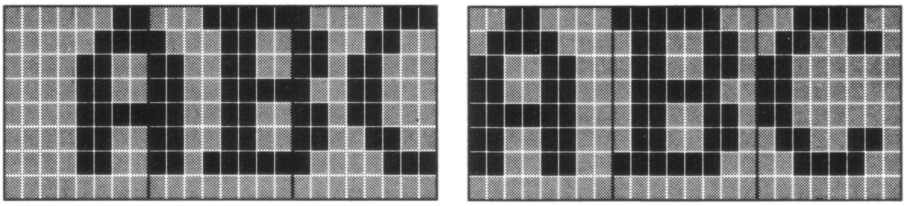
Il codice sorgente dei due tipi di routine di testo grafico è simile. Gli esempi di questo capitolo applicano il secondo metodo, questo metodo rende però le routine più complicate di quelle che tracciano solo i pixel di primo piano. Potete convertire le routine in modo che traccino solo i pixel di primo piano eliminando il codice per l'inglobamento dei pixel di sfondo.

Progettazione di un generatore di caratteri software

I generatori di caratteri software per i sistemi di visualizzazione dei PC IBM presentano diverse problematiche comuni di progettazione. Dal momento che le prestazioni del vostro generatore di caratteri influenzano fortemente le prestazioni generali di molte applicazioni grafiche, è necessario sempre valutare i compromessi tra funzionalità e semplicità delle vostre routine di generazione di caratteri.

Allineamento orizzontale

Nelle modalità grafiche, il bordo sinistro di un carattere non è necessariamente allineato al byte. Quando un carattere viene scritto in modo che i suoi pixel all'estrema sinistra ricadano più o meno al centro di un byte nel buffer del video (vedere Figura 96a), il generatore di caratteri deve spostare e mascherare la matrice di carattere in modo che vengano aggiornati soltanto i pixel che fanno parte del carattere.



a. b.

Figura 96. *Allineamento dei caratteri nel buffer del video. Nella Figura 96a, i caratteri non sono allineati; nella Figura 96b, i caratteri sono invece allineati al byte*

Di solito, però, i caratteri vengono scritti nel buffer del video ad indirizzi di pixel allineati ai byte (vedere Figura 96b). Questo avviene, ad esempio, quando il monitor viene utilizzato in "modalità telescrivente", cioè quando ogni riga di caratteri inizia dal margine sinistro dello schermo. La produzione di caratteri allineati al byte non richiede né rotazione né mascheramento di pixel, quindi l'uso di una routine separata per i caratteri allineati al byte migliora le prestazioni del generatore di caratteri.

Dimensioni variabili dei caratteri

La scrittura di un generatore di caratteri che permetta la visualizzazione di caratteri di diverse altezze è un compito relativamente facile. L'altezza di un carattere corrisponde al numero di byte presente nella propria definizione all'interno della tabella di definizione caratteri. Potete quindi utilizzare l'altezza dei vostri caratteri come limite di loop all'interno della routine di generatore di caratteri senza influenzare significativamente la routine nel suo complesso.

La gestione di caratteri di diverse larghezze si presenta più difficile. Se la larghezza di un carattere non corrisponde esattamente ad un numero intero di byte, dovrete mascherare ogni riga di pixel del carattere quando lo memorizzate nel buffer del video. Anche in questo caso, le operazioni di servizio supplementari per la produzione della maschera di bit appropriata e per il mascheramento dei pixel nel buffer del video complicano e rallentano la routine di generatore di caratteri.

Definizione

Potete definire i caratteri in diversi modi. Il metodo più semplice è rappresentato dalla definizione dell'intero carattere prima di memorizzarlo nel buffer del video; se una qualsiasi parte della matrice del carattere ricadrà all'esterno dell'area definita, non scrivete il carattere.

La definizione di un carattere in modo che solo una parte di esso venga memorizzata nel buffer del video è più difficile. Un metodo per produrre questo risultato è quello di modificare il generatore di caratteri in modo che una qualsiasi parte definita di un carattere non venga scritta nel buffer. Un altro approccio è quello di scrivere l'intero carattere in un buffer ausiliario e quindi di copiare il carattere definito nel buffer del video con una routine di copiatura di blocchi di pixel (vedere Capitolo 11).

Orientamento dei caratteri

Di solito i caratteri vengono visualizzati in modo che possano essere letti da sinistra verso destra e dall'alto al basso. Per modificare questo orientamento, applicate la trasformazione appropriata alle configurazioni di bit nella tabella di definizione caratteri. Ad esempio, la subroutine del Listato 82 ruota la configurazione di bit a 8 byte che rappresenta un carattere 8 per 8 in modo che i caratteri visualizzati vengano letti dal basso verso l'alto. Con questa trasformazione, potete usare lo stesso generatore di caratteri per visualizzare caratteri orientati verticalmente o orizzontalmente. Vengono modificate solo le configurazioni di bit.

```
            mov     si,seg OldCharDef
            mov     ds,si
            mov     si,offset OldCharDef        ; DS:SI -> definizione
                                                ;  vecchio carattere

            mov     di,seg NewCharDef
            mov     es,di
            mov     di,offset NewCharDef        ; ES:DI -> definizione
                                                ;  nuovo carattere

            mov     bx,1            ; BH := 0
                                    ; BL := maschera di bit

>L01:       push    si              ; mantiene SI
            mov     cx,8            ; CX := numero di bit di ogni byte

L02:        lodsb                   ; AL := byte successivo della
                                    ;  definizione del vecchio carattere
            and     al,bl           ; maschera un bit
            cmp     bh,al           ; imposta flag del riporto se bit
                                    ;  di maschera diverso da zero
```

```
rcl     ah,1            ; ruota bit in AH
loop    L02             ; loop su definizione vecchio carattere

mov     al,ah


stosb                   ; memorizza byte successivo della
                        ;  definizione del nuovo carattere
pop     si              ; DS:SI -> inizio della definizione
                        ;  del vecchio carattere
shl     bl,1            ; BL := nuova maschera di bit
jnz     L01             ; loop fino a che maschera di bit viene
                        ;  spostata fuori da BL
```

Listato 82. *Una routine che ruota di 90 gradi una definizione di carattere 8 per 8.*

Collaborazione con il BIOS del video

Anche se le vostre tabelle di definizione caratteri e il software del generatore di caratteri evitano l'uso delle funzioni del BIOS del video, dovreste in ogni caso, quando possibile, tentare di mantenere la compatibilità collaborando con le routine del BIOS. Nelle modalità grafiche a 200 linee, dovreste aggiornare l'indirizzo del vettore di interrupt 1FH ogni volta che utilizzate una tabella di definizione carattere 8 per 8 che comprenda i secondi 128 caratteri ASCII. Su EGA, VGA e MCGA dovreste in generale utilizzare la funzione 11H di INT 10H per mantenere aggiornati i vettori di interrupt del BIOS e le variabili dell'area dati di visualizzazione.

Più potenza, più complessità

Potete aggiungere funzionalità ad un generatore di caratteri software in diversi modi. Potreste, ad esempio, scrivere un generatore di caratteri che si riferisca ad una tabella di larghezze relative di caratteri per la visualizzazione di caratteri a spaziatura proporzionale. Mano a mano che la vostra routine legge le configurazioni di bit dalla tabella di definizione caratteri, potreste dover fare in modo che la routine le sposti a destra di un numero predeterminato di pixel per generare set di caratteri in grassetto o in corsivo. Potreste applicare una configurazione di valori di pixel ai pixel di primo piano aggiornati. Potreste far sì che una tabella di definizione caratteri si estenda oltre la solita gamma di 256 caratteri; più caratteri definite, più ampia sarà la gamma dei caratteri visualizzabili contemporaneamente. Ognuna di queste possibilità aggiunge potenza e flessibilità al vostro generatore di caratteri software, ma tutte queste possibilità complicano il codice sorgente e alla fine lo rallentano.

Implementazione di un generatore di caratteri software

Tutti gli esempi di generatori di caratteri software contenuti in questo capitolo richiedono la specifica delle coordinate *x* e *y* del pixel situato nell'angolo superiore sinistro della matrice di carattere visualizzato. Ogni routine rileva il caso particolare in cui la matrice di carattere è allineata al byte nel buffer del video, ma le routine non verificano le coordinate di pixel né eseguono alcuna definizione. Tutte le routine tranne *DisplayChar10()* aggiornano i pixel nel buffer del video sostituendo i loro valori. Per eseguire un'operazione logica AND, OR o XOR, dovete modificare le routine (vedere Capitolo 5).

CGA

Nella modalità a 2 colori 640 per 200 su CGA, il generatore di caratteri software applica le configurazioni di bit della tabella di definizione caratteri direttamente ai pixel nel buffer del video (vedere Listato 83). Quando il carattere è allineato al byte nel buffer del video, la routine copia i valori di pixel direttamente dalla tabella di definizione caratteri. In caso contrario, per ogni riga di otto pixel del carattere, viene utilizzata una maschera di 16 bit ruotata per azzerare gli appropriati otto pixel nel buffer. Successivamente, i pixel della tabella di definizione caratteri vengono ruotati in posizione e memorizzati nel buffer utilizzando un'operazone logica OR.

```
                TITLE   'Listato 83'
                NAME    DisplayChar06
                PAGE    55,132

;
; Nome:         DisplayChar06
;
; Funzione:     Visualizza un carattere nella modalità a 2 colori 640x200
;
; Chiamante:    Microsoft C:
;
;                       azzera DisplayChar06(c,x,y,fgd,bkgd);
;
;                       int c;                  /* codice carattere */
;
;                       int x,y;                /* pixel superiore sinistro */
;
;                       int fgd,bkgd;           /* valori di pixel
;                                                  di primo piano e di sfondo */
;
```

```
ARGc            EQU     word ptr [bp+4] ; indirizzamento di cornice di stack
ARGx            EQU     word ptr [bp+6]
ARGy            EQU     word ptr [bp+8]
ARGfgd          EQU     byte ptr [bp+10]
ARGbkgd         EQU     byte ptr [bp+12]

VARmask         EQU     [bp-2]
VARtoggle       EQU     [bp-4]

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr06:near

                PUBLIC _DisplayChar06
_DisplayChar06PROC      near

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                sub     sp,4        ; spazio di stack per variabili locali
                push    si
                push    di
                push    ds

; imposta maschera di commutazione pixel di primo piano

                mov     ah,ARGfgd   ; AH := 0 o 1 (valore di pixel di primo
                                    ;  piano)
                ror     ah,1        ; bit più significativo di AH := 0 o 1
                cwd                 ; estende bit più significativo a tutto DX
                not     dx          ; DX :=    0 se primo piano = 1
                                    ;    o FFFFh se primo piano = 0
                mov     VARtoggle,dx

; calcola indirizzo primo pixel

                mov     ax,ARGy     ; AX := y
                mov     bx,ARGx     ; BX := x
                call    PixelAddr06; ES:BX - buffer
                                    ; CL := # bit da spostare a sinistra

                xor     cl,7        ; CL := # bit da ruotare a destra

                mov     ax,0FF00h
                ror     ax,cl       ; AX := maschera di bit nella posizione
                                    ;  corretta
                mov     VARmask,ax

; imposta indirizzamento del buffer del video

                mov     dx,2000h    ; incremento per intercal.buffer del video
                mov     di,80-2000h; incremento dall'ultima alla prima
                                    ;  intercalazione

                test    bx,2000h    ; imposta flag zero se BX è nella prima
                                    ;  intercalazione
```

```
            jz      L01

            xchg    di,dx       ; scambia valori di incremento se primo
                                ;  pixel si trova nella prima intercal.

; imposta indirizzamento tabella definizione caratteri

L01:        push    bx          ; mantiene indirizzo buffer


            mov     ax,40h
            mov     ds,ax       ; DS := segmento dell'area dati di
                                ;  visualizzazione del BIOS
            mov     ch,ds:[85h] ; CH := POINTS (righe di pixel nel carat.)

            xor     ax,ax
            mov     ds,ax       ; DS := zero assoluto

            mov     ax,ARGc     ; AL := codice di carattere
            cmp     al,80h
            jae     L02

            mov     bx,43h*4    ; DS:BX -> vettore int 43h se car.< 80h
            jmp     short L03

L02:        mov     bx,1Fh*4    ; DS:BX -> vettore int 1Fh se car.> = 80h
            sub     al,80h      ; introduce codice carattere nella gamma
                                ;  di tabella

L03:        lds     si,ds:[bx]  ; DS:SI -> inizio della tabella di caratteri
            mul     ch          ; AX := offset nella tabella definizione
                                ;  caratteri
                                ;  (POINTS * codice carattere)
            add     si,ax       ; SI := indirizzo di definizione carattere

            pop     bx          ; ripristina indirizzo buffer

            test    cl,cl       ; test # bit da ruotare
            jnz     L20         ; salta se carattere non è allineato al
                                ;  byte


; routine per caratteri allineati al byte

            mov     ah,VARtoggle    ; AH := maschera di commutazione primo
                                    ;  piano
            xchg    ch,cl       ; CX := POINTS

L10:        lodsb               ; AL := configurazione di bit per riga di
                                ; pixel successiva
            xor     al,ah       ; commuta pixel se primo piano = 0
            mov     es:[bx],al  ; memorizza pixel nel buffer

            add     bx,dx       ; BX := riga successiva nel buffer
            xchg    di,dx       ; scambia incrementi di buffer
```

```
            loop    L10
            jmp     short Lexit


; routine per caratteri non allineati al byte

L20:        mov     ax,VARmask
            and     es:[bx],ax ; maschera pixel del carattere nel buffer

            xor     ah,ah
            lodsb              ; AX := configurazione di bit per riga di
                               ;  pixel successiva
            xor     al,VARtoggle   ; commuta pixel se primo piano = 0

            ror     ax,cl      ; ruota pixel in posizione
            or      es:[bx],ax ; memorizza pixel nel buffer

            add     bx,dx      ; BX := riga successiva nel buffer
            xchg    di,dx      ; scambia incrementi del buffer
            dec     ch
            jnz     L20


Lexit:      pop     ds         ; ripristina registri e ritorna
            pop     di
            pop     si
            mov     sp,bp
            pop     bp
            ret

_DisplayChar06ENDP

_TEXT       ENDS

            END
```

Listato 83. *Un generatore di caratteri software per la modalità a 2 colori 640 per 200.*

La routine per la modalità a 4 colori 320 per 200 del Listato 84 è molto più complicata in quanto ogni bit della definizione carattere deve essere espanso per essere trasformato nell'appropriato valore di pixel a 2 bit. Un bit 0 nella tabella di definizione caratteri diventa un valore di pixel di sfondo a 2 bit; un bit 1 nella tabella viene trasformato in un valore di pixel di primo piano a 2 bit. Di conseguenza, ogni byte della tabella viene trasformato in una word di pixel.

```
                TITLE   'Listato 84'
                NAME    DisplayChar04
                PAGE    55,132

;
; Nome:         DisplayChar04
;
; Funzione:     Visualizza un carattere nella modalità grafica a 4 colori
;320x200
;
; Chiamante:    Microsoft C:
;
;                       azzera DisplayChar04(c,x,y,fgd,bkgd);
;
;                       int c;                  /* codice di carattere */
;
;                       int x,y;                /* pixel superiore sinistro */
;
;                       int fgd,bkgd;           /* valori di pixel
;                                                  di primo piano e di sfondo */
;

ARGc            EQU     word ptr [bp+4] ; indirizzamento di cornice di stack
ARGx            EQU     word ptr [bp+6]
ARGy            EQU     word ptr [bp+8]
ARGfgd          EQU              [bp+10]
ARGbkgd         EQU              [bp+12]

VARshift        EQU     word ptr [bp-2]
VARincr         EQU     word ptr [bp-4]


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr04:near

                PUBLIC _DisplayChar04
_DisplayChar04PROC      near

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                sub     sp,4        ; spazio di stack per variabili locali
                push    si
                push    di
                push    ds
; estende valori di pixel

                mov     bx,offset DGROUP:PropagatedPixel
                mov     al,ARGfgd
                xlat                ; estende valore di pixel di primo piano
                mov     ah,al
                mov     ARGfgd,ax
                mov     al,ARGbkgd
```

```
            xlat                ; estende valore di pixel di sfondo
            mov     ah,al
            mov     ARGbkgd,ax

; calcola indirizzo primo pixel

            mov     ax,ARGy     ; AX := y
            mov     bx,ARGx     ; BX := x
            call    PixelAddr04; ES:BX - > buffer
                                ; CL := # bit da spostare a sinistra
                                ;  per mascherare il pixel
            mov     ch,0FCh
            shl     ch,cl       ; CH := maschera di bit per lato destro
                                ;  del carattere

            xor     cl,6        ; CL := 6 - CL (# bit per ruotare
                                ;  carattere in posizione)
            mov     VARshift,cx

; imposta indirizzamento buffer del video

            mov     di,2000h    ; incremento per interc. buffer del video
            mov     VARincr,80-2000h      ; incremento da ultima a prima
                                          ;  intercalazione

            test    bx,2000h    ; imposta flag zero se BX è nella prima
                                ;  intercalazione
            jz      L01

            xchg    VARincr,di  ; scambia valori di incremento se primo
                                ;  pixel si trova nella prima intercal.

; imposta indirizzamento tabella definizione caratteri

L01:        push    bx          ; mantiene indirizzo buffer

            mov     ax,40h
            mov     ds,ax       ; DS := segmento dell'area dati di
                                ;  visualizzazione del BIOS
            mov     ch,ds:[85h]; CH := POINTS (righe di pixel nel
                                ;  carattere)

            xor     ax,ax
            mov     ds,ax       ; DS := zero assoluto

            mov     ax,ARGc     ; AL := codice carattere
            cmp     al,80h
            jae     L02

            mov     bx,43h*4    ; DS:BX -> vettore int 43h se car. < 80h
            jmp     short L03

L02:        mov     bx,1Fh*4    ; DS:BX -> vettore int 1Fh se car. >= 80h
            sub     al,80h      ; introduce codice carattere nella gamma
                                ;  di tabella

L03:        lds     si,ds:[bx]  ; DS:SI -> inizio tabella dei caratteri
```

```
                mul     ch          ; AX := offset nella tabella definizione
                                    ;  caratteri
                                    ;  (POINTS * codice carattere)
                add     si,ax       ; SI := indirizzo definizione  carattere

                pop     bx          ; ripristina indirizzo buffer

                xchg    ch,cl       ; CH := # bit da ruotare
                                    ; CL := POINTS

                test    ch,ch       ; test # bit da ruotare
                jnz     L20         ; salta se carattere non è allineato al
                                    ;  byte
; routine per caratteri allineati al byte

L10:            lodsb               ; AL := configurazione di bit per riga
                                    ;  successiva di pixel
                xor     dx,dx       ; DX := valore iniziale dei bit
                                    ;raddoppiati
                mov     ah,8        ; AH := # di bit nella configurazione

L11:            shr     al,1        ; cf := bit meno significativo di AL
                rcr     dx,1        ; bit più significativo di CX := cf
                sar     dx,1        ; raddoppia bit più significativi di DX
                dec     ah          ; loop 8 volte
                jnz     L11

                mov     ax,dx       ; AX,DX := configuraz. di bit raddoppiati
                and     ax,ARGfgd   ; AX := pixel di primo piano
                not     dx
                and     dx,ARGbkgd  ; DX := pixel di sfondo

                or      ax,dx       ; AX := otto pixel
                xchg    ah,al       ; introduce byte nell'ordine corretto
                mov     es:[bx],ax  ; aggiorna buffer del video

                add     bx,di       ; BX := riga successiva nel buffer
                xchg    di,VARincr  ; scambia incrementi del buffer

                loop    L10
                jmp     short Lexit


; routine per caratteri non allineati al byte

L20:            xor     ch,ch       ; CX := POINTS

L21:            push    cx          ; mantiene CX

                mov     cx,VARshift ; CH := maschera per lato destro del
                                    ;  carattere
                                    ; CL := # bit da ruotare

                lodsb               ; AL := configurazione di bit per riga
                                    ;  successiva di pixel
                xor     dx,dx       ; DX := valore iniziale dei bit
                                    ;  raddoppiati
```

```
            mov     ah,8        ; AH := # di bit nella configurazione

L22:        shr     al,1        ; DX := raddoppia bit in AL
            rcr     dx,1        ;  (come sopra)
            sar     dx,1
            dec     ah
            jnz     L22

            xchg    dh,dl       ; DH := bit per metà destra del carattere
                                ; DL := bit per metà sinistra del
                                ;  carattere
            mov     ax,dx

            and     ax,ARGfgd   ; AX := pixel di primo piano
            not     dx
            and     dx,ARGbkgd  ; DX := pixel di sfondo

            or      dx,ax       ; DX := otto pixel
            ror     dx,cl       ; DH := pixel del lato destro e sinistro
                                 DL := pixel di centro
            mov     al,ch
            xor     ah,ah       ; AX := maschera dei byte di sinistra e di
                                ;  centro del carattere
            and     es:[bx],ax  ; azzera pixel nel buffer del video

            not     ax
            and     ax,dx
            or      es:[bx],ax  ; aggiorna pixel nei byte di sinistra e
                                ;  del centro

            mov     al,ch       ; AL := maschera del byte di destra
            not     al
            and     es:[bx+2],al; maschera pixel nel byte di destra
                                 ;  nel buffer
            and     ch,dl
            or      es:[bx+2],ch; aggiorna pixel nel byte di destra

            add     bx,di       ; BX := riga successiva nel buffer
            xchg    di,VARincr  ; scambia incrementi del buffer

            pop     cx          ; ripristina CX
            loop    L21
Lexit:      pop     ds          ; ripristina registri e ritorna
            pop     di
            pop     si
            mov     sp,bp
            pop     bp
            ret

_DisplayChar04ENDP

_TEXT       ENDS


_DATA       SEGMENTword public 'DATA'
```

```
PropagatedPixel        DB          00000000b ; 0
                       DB          01010101b ; 1
                       DB          10101010b ; 2
                       DB          11111111b ; 3

_DATA          ENDS

               END
```

Listato 84. *Un generatore di caratteri software per la modalità a 4 colori 320 per 200.*

Nel Listato 84, quando il carattere è allineato al byte nel buffer del video, la routine sposta la word di 16 bit dei pixel direttamente nel buffer. Un carattere non allineato al byte si estende per tre byte nel buffer. In questo caso, la routine deve ruotare in posizione gli otto pixel di ogni riga del carattere. Successivamente, i primi due byte del carattere nel buffer vengono mascherati ed aggiornati, seguiti dal terzo byte (posto all'estrema destra) del carattere.

HGC e HGC+

Una routine per la modalità grafica monocromatica 720 per 348 su HGC e HGC+ può utilizzare la stessa tecnica di mascheramento di bit utilizzata dalla routine per la modalità a 2 colori 640 per 200 CGA. Potete convertire *DisplayChar06()* in una routine compatibile Hercules correggendo la chiamata a *PixelAddr06()* e modificando l'indirizzamento del buffer del video in modo che supporti la diversa struttura del buffer video dei due adattatori.

Vale la pena, in ogni caso, sfruttare la risoluzione orizzontale di 720 pixel dell'HGC visualizzando caratteri in una matrice larga 9 pixel, in modo che ogni riga sullo schermo contenga 80 caratteri equamente spaziati. La routine del Listato 85 realizza ciò aggiungendo un nono bit ad ogni configurazione di 8 bit letta dalla tabella di definizione caratteri. Il nono bit è 0 tranne che nel caso di caratteri di tracciamento riquadri (codici ASCII da 0C0 a 0DFH). Per quanto riguarda questi caratteri, il nono bit è una copia del bit posto all'estrema detra nella configurazione di bit (questo imita la funzione del generatore di caratteri hardware nelle modalità alfanumeriche. Vedere Capitolo 10).

```
               TITLE   'Listato 85'
               NAME    DisplayCharHGC
               PAGE    55,132

;
; Nome:        DisplayCharHGC
;

; Funzione:    Visualizza un carattere nella modalità grafica monocromatica
;              720x348 Hercules
;
```

```
; Chiamante:  Microsoft C:
;
;                    azzera DisplayCharHGC(c,x,y,fgd,bkgd);
;
;                    int c;                /* codice carattere */
;
;                    int x,y;              /* pixel superiore sinistro */
;
;                    int fgd,bkgd;         /* valori di pixel
;                                             di primo piano e di sfondo */
;

ARGc          EQU    word ptr [bp+4] ; indirizzamento di cornice di stack
ARGx          EQU    word ptr [bp+6]
ARGy          EQU    word ptr [bp+8]
ARGfgd        EQU    byte ptr [bp+10]
ARGbkgd       EQU    byte ptr [bp+12]

VARmask       EQU             [bp-2]
VARtoggle     EQU             [bp-4]
VAR9bits      EQU    byte ptr [bp-6]


_TEXT         SEGMENTbyte public 'CODE'
              ASSUME cs:_TEXT

              EXTRN  PixelAddrHGC:near

              PUBLIC _DisplayCharHGC
_DisplayCharHGC      PROC       near

              push   bp         ; mantiene registri del chiamante
              mov    bp,sp
              sub    sp,6       ; spazio di stack per variabili locali
              push   si
              push   di
              push   ds

; calcola indirizzo primo pixel

              mov    ax,ARGy    ; AX := y
              mov    bx,ARGx    ; BX := x
              call   PixelAddrHGC   ; ES:BX -> buffer
                                ; CL := # bit da spostare a sinistra

              xor    cl,7       ; CL := # bit da ruotare a destra

; imposta maschera a 8 o 9 bit

              mov    ax,40h
              mov    ds,ax      ; DS := segmento dell'area dati
                                ;  di visualizzazione del BIOS
              mov    ax,0FF0h   ; AX := maschera a 8 bit
              mov    VAR9bits,0 ; azzera questo flag

              cmp    byte ptr ds:[4Ah],90 ; CRT_COLS = 90?
              je     L01        ; salta se caratteri sono larghi 8 pixel
```

```
            mov     ah,7Fh      ; AX :=maschera a 9 bit
            cmp     ARGc,0C0h
            jb      L01         ; salta se codice carattere ...

            cmp     ARGc,0DFh
            ja      L01         ; ... è al di fuori della gamma 0C0-0DFh

            inc     VAR9bits    ; imposta flag per estendere a 9 bit

L01:        ror     ax,cl       ; AX := maschera di bit nella posizione
                                ;  corretta
            mov     VARmask,ax

; imposta maschera di commutazione pixel di primo piano

            mov     ah,ARGfgd   ; AH := 0 o 1 (valore di pixel di primo
                                ;  piano)
            ror     ah,1        ; bit più significativo di AH := 0 o 1
            cwd                 ; estende bit più significativo a tutto DX
            not     dx          ; DX := 0 se primo piano = 1
                                ;       o FFFh se primo piano = 0
            mov     ax,VARmask
            not     ax
            and     dx,ax       ; azzera bit non utiliz. della maschera di
                                ;  commutaz. di DX
            mov     VARtoggle,dx

; imposta indirizzamento tabella definizione caratteri

            push    bx          ; mantiene indirizzo buffer

            mov     ch,ds:[85h]; CH := POINTS (righe di pixel nel carat.)

            xor     ax,ax
            mov     ds,ax       ; DS := zero assoluto

            mov     ax,ARGc     ; AL := codice di carattere
            cmp     al,80h
            jae     L02

            mov     bx,43h*4    ; DS:BX -> vettore int 43h se car.< 80h
            jmp     short L03

L02:        mov     bx,1Fh*4    ; DS:BX -> vettore int 1Fh se car. >= 80h
            sub     al,80h      ; introduce codice di carattere nella
                                ;  gamma di tabella

L03:        lds     si,ds:[bx]  ; DS:SI -> inizio tabella di caratteri
            mul     ch          ; AX := offset nella tabella definiz. car.
                                ;POINTS * codice carattere)
            add     si,ax       ; SI := indirizzo di definizione carattere

            pop     bx          ; ripristina indirizzo buffer

; maschera e imposta pixel nel buffer del video

L20:        mov     ax,VARmask
            and     es:[bx],ax  ; maschera pixel del carattere nel buffer
```

```
            xor     ah,ah
            lodsb                   ; AX := configurazione di bit per riga
                                    ;  successiva di pixel
            cmp     VAR9bits,0
            je      L21             ; salta se il carattere è largo 8 pixel


            ror     ax,1            ; copia bit meno significativo di AX nel..
            rcl     al,1            ;  bit più significativo

L21:        ror     ax,cl           ; ruota pixel in posizione
            xor     ax,
                    VARtoggle       ; commuta pixel se primo piano = 0
            or      es:[bx],ax      ; memorizza pixel nel buffer

            add     bx,2000h        ; incrementa ad area succ. dell'intercal.
            jns     L22

            add     bx,90-8000h; incrementa alla prima area di intercal.

L22:        dec     ch
            jnz     L20
Lexit:      pop     ds              ; ripristina registri e ritorna
            pop     di
            pop     si
            mov     sp,bp
            pop     bp
            ret

_DisplayChar06ENDP

_TEXT       ENDS

            END
```

Listato 85. *Un generatore di caratteri software per la modalità grafica monocromatica Hercules.*

CONSIGLIO

Si noti come le routine per CGA e per Hercules utilizzino il vettore di interrupt 43H per puntare l'inizio della tabella corrente di definizione caratteri. Questo è il vettore di interrupt che il BIOS ROM dell'EGA e della VGA utilizza per questo scopo. Inoltre, le routine determinano la dimensione della matrice di carattere visualizzato esaminando le variabili POINTS (0040:0085) e CRT_COLS (0040:004A) nell'area dati di visualizzazione del BIOS. Se non state utilizzando un EGA, un'MCGA o una VGA, il BIOS non terrà aggiornato il vettore di interrupt e POINTS; in questo caso, il vostro programma dovrebbe o aggiornare questi valori in modo esplicito o mantenere valori equivalenti da qualche altra parte.

MCGA

Nella modalità a 2 colori 640 per 480 dell'MCGA, i pixel vengono memorizzati a gruppi di otto per ogni byte, quindi potete adattare il generatore di caratteri per la modalità a 2 colori 640 per 200 per l'uso in questa modalità modificando il suo indirizzamento di buffer. Un generatore di caratteri per la modalità a 256 colori 320 per 200 è leggermente diverso, in quanto ogni bit della tabella di definizione carattere si espande in un byte all'interno del buffer del video (vedere Listato 86).

```
                TITLE   'Listato 86'
                NAME    DisplayChar13
                PAGE    55,132

;
; Nome:         DisplayChar13
;
; Funzione:     Visualizza un carattere nella modalità a 256 colori 320x200
;               MCGA/VGA
;
; Chiamante:    Microsoft C:
;
;                       azzera DisplayChar13(c,x,y,fgd,bkgd);
;
;                       int c;                  /* codice di carattere */
;
;                       int x,y;                /* pixel superiore sinistro */
;
;                       int fgd,bkgd;           /* valori di pixel
;                                                  di primo piano e di sfondo */
;

ARGc            EQU     word ptr [bp+4] ; indirizzamento di cornice di stack
ARGx            EQU     word ptr [bp+6]
ARGy            EQU     word ptr [bp+8]
ARGfgd          EQU     byte ptr [bp+10]
ARGbkgd         EQU     byte ptr [bp+12]

BytesPerLine    EQU     320

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr13:near

                PUBLIC _DisplayChar13
_DisplayChar13PROC      near

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                push    si
                push    di
                push    ds
```

```
; calcola primo indirizzo di pixel

            mov     ax,ARGy     ; AX := y
            mov     bx,ARGx     ; BX := x
            call    PixelAddr13; ES:BX -> buffer
            mov     di,bx       ; ES:DI -> buffer

; imposta indirizzamento tabella definizione carattere

            mov     ax,40h
            mov     ds,ax       ; DS := segmento dell'area dati
                                ;  di visualizzazione del BIOS
            mov     cx,ds:[85h]; CX := POINTS (righe di pixel nel carat.)

            xor     ax,ax
            mov     ds,ax       ; DS := zero assoluto

            mov     ax,ARGc     ; AL := codice di carattere
            mov     bx,43h*4    ; DS:BX -> vettore int 43h se car. < 80h
            lds     si,ds:[bx]  ; DS:SI -> inizio tabella dei caratteri
            mul     cl          ; AX := offset nella tabella di definizione
                                ;  car.(POINTS * codice carattere)
            add     si,ax       ; SI := indirizzo della definiz. carattere
; memorizza il carattere nel buffer del video

            mov     bl,ARGfgd   ; BL := valore di pixel di primo piano
            mov     bh,ARGbkgd  ; BH := valore di pixel di sfondo

L10:        push    cx          ; mantiene CX per tutto il loop
            mov     cx,8        ; CX := larghezza carattere in pixel
            lodsb
            mov     ah,al       ; AH := configurazione di bit per
                                ;  successiva riga di pixel

L11:        mov     al,bl       ; AL := valore di pixel di primo piano
            shl     ah,1        ; flag del riporto:= bit più significativo
            jc      L12         ; salta se configurazione di bit specifica
                                ;  un pixel di primo piano (bit = 1)
            mov     al,bh       ; AL := valore di pixel di sfondo

L12:        stosb               ; aggiorna un pixel nel buffer
            loop    L11

            add     di,BytesPerLine-8     ; incrementa indirizzo di buffer
                                          ;  a successiva riga di pixel
            pop     cx
            loop    L10         ; loop del carattere verso il basso

            pop     ds          ; ripristina registri e ritorna
            pop     di
            pop     si
            mov     sp,bp
            pop     bp
            ret
```

```
_DisplayChar13ENDP

_TEXT           ENDS
                END
```

Listato 86. *Un generatore di caratteri per la modalità a 256 colori 320 per 200 dell'MCGA e della VGA.*

EGA e VGA

La routine per EGA e VGA del Listato 87 usa il controller grafico per aggiornare i pixel nel buffer del video. La routine è analoga sotto certi aspetti alla routine per la modalità a 2 colori 640 per 200 del CGA, in quanto ogni byte del buffer del video rappresenta otto pixel. Naturalmente, il codice viene complicato dalla necessità di programmare il controller grafico in modo che gestisca i valori di pixel di primo piano e di sfondo.

La routine scrive ogni riga di pixel del carattere creando dei blocchi dei piani di bit, aggiornando i pixel di primo piano, aggiornando i pixel di sfondo e quindi riscrivendo i blocchi nei piani di bit. Il controller grafico non può facilmente aggiornare contemporaneamente sia i pixel di primo piano sia quelli di sfondo, quindi la routine deve eseguire separatamente queste operazioni.

```
                TITLE   'Listato 87'
                NAME    DisplayChar10
                PAGE    55,132

;
; Nome:         DisplayChar10
;
; Funzione:     Visualizza un carattere nelle modalità grafiche EGA e VGA
;               originali
;
; Chiamante:    Microsoft C:
;
;                       azzera DisplayChar10(c,x,y,fgd,bkgd);
;
;                       int c;                  /* codice di carattere */
;
;                       int x,y;                /* pixel superiore sinistro */
;
;                       int fgd,bkgd;           /* valori di pixel
;                                                  di primo piano e di sfondo */
;

ARGc            EQU     word ptr [bp+4] ; indirizzamento di cornice di stack
ARGx            EQU     word ptr [bp+6]
ARGy            EQU     word ptr [bp+8]
ARGfgd          EQU     byte ptr [bp+10]
ARGbkgd         EQU     byte ptr [bp+12]

VARshift        EQU             [bp-2]
```

```
BytesPerLine  =       80          ; (deve essere 40 nella modalità a 16
                                  ;  colori 320x200)
RMWbits       =       18h         ; legge-modifica-scrive  bit

_TEXT         SEGMENTbyte public 'CODE'
              ASSUME cs:_TEXT

              EXTRN  PixelAddr10:near

              PUBLIC _DisplayChar10
_DisplayChar10PROC   near

              push   bp          ; mantiene registri del chiamante
              mov    bp,sp
              sub    sp,2        ; spazio di stack per variabile locale
              push   si
              push   di
              push   ds

; calcola indirizzo del primo pixel

              mov    ax,ARGy     ; AX := y
              mov    bx,ARGx     ; BX := x
              call   PixelAddr10; ES:BX -> buffer
                                 ; CL := # bit da spostare a sinistra
                                 ;  per mascherare pixel
              inc    cx
              and    cl,7        ; CL := # bit da spostare per
                                 ; mascherare carattere

              mov    ch,0FFh
              shl    ch,cl       ; CH := maschera di bit per lato destro
                                 ; del carattere
              mov    VARshift,cx

              push   es          ; mantiene segmento del buffer del
                                 ;  video
              mov    si,bx       ; SI := offset del buffer del video

; imposta indirizzamento tabella definizione carattere

              mov    ax,40h
              mov    ds,ax       ; DS := segmento dell'area dati
                                 ;  di visualizzazione del BIOS
              mov    cx,ds:[85h]; CX := POINTS (righe di pixel nel
                                 ;  carattere)

              xor    ax,ax
              mov    ds,ax       ; DS := zero assoluto
              mov    ax,ARGc     ; AL := codice di carattere
              mov    bx,43h*4    ; DS:BX -> vettore int 43h
              les    di,ds:[bx]  ; ES:DI -> inizio tabella dei caratteri
              mul    cl          ; AX := offset nella tabella di
                                 ;  definizione caratteri
                                 ;  (POINTS * codice del carattere)
              add    di,ax       ; DI := indirizzo definiz. carattere
```

```
            pop     ds          ; DS:SI - buffer del video

; imposta registri del controller grafico

            mov     dx,3CEh     ; porta reg indirizzo controller
                                ;  grafico

mov         ax,0A05h            ; AL :=  numero registro modalità
                                ; AH :=  modalità scrittura 2 (bit 0-1)
                                ;        modalità lettura 1 (bit 4)
            out     dx,ax

            mov     ah,RMWbits  ; AH := legge-modifica-scrive bit
            mov     al,3        ; AL := reg rotaz. dati/selez. funzione
            out     dx,ax

            mov     ax,0007     ; AH := bit colore ininfluente
                                ; AL := numero reg colore ininfluente
            out     dx,ax       ; "ininfluente" per tutti  piani di bit

; seleziona routine di output a seconda che il carattere sia allineato o
; meno al byte

            mov     bl,ARGfgd   ; BL := valore di pixel di primo piano
            mov     bh,ARGbkgd  ; BH := valore di pixel di sfondo

            cmp     byte ptr VARshift,0  ; test # bit da spostare
            jne     L20         ; salta se carattere non è allineato al
                                ;  byte


; routine per caratteri allineati al byte

            mov     al,8        ; AL := numero registro maschera di bit

L10:        mov     ah,es:[di]  ; AH := configurazione per riga
                                ;  successiva di pixel
            out     dx,ax       ; aggiorna registro di maschera di bit

            and     [si],bl     ; aggiorna pixel di primo piano

            not     ah
            out     dx,ax
            and     [si],bh     ; aggiorna pixel di sfondo

            inc     di          ; ES:DI -> byte success. nella tabella
                                ;  definizione car.
            add     si,Bytes
                    PerLine     ; incrementa a linea successiva nel
                                ;  buffer del video
            loop    L10
            jmp     short Lexit
; routine per caratteri non allineati al byte

L20:        push    cx          ; mantiene contatore di loop

            mov     cx,VARshift; CH := maschera per lato sinistro del
                                ;  carattere
```

```
                                ; CL := # bit da spostare a sinistra
; lato sinistro del carattere


            mov     al,es:[di]  ; AL := bit per riga succes. di pixel
            xor     ah,ah
            shl     ax,cl       ; AH := bit per lato sinistro del car.
                                ; AL := bit per lato destro del carat.
            push    ax          ; salva bit per lato destro su stack
            mov     al,8        ; AL := numero registro maschera di bit
            out     dx,ax       ; imposta maschera di bit per pixel di
                                ;  primo piano

            and     [si],bl     ; aggiorna pixel di primo piano

            not     ch          ; CH := maschera per lato sinistro del
                                ;  carattere
            xor     ah,ch       ; AH := bit per pixel di sfondo
            out     dx,ax       ; imposta maschera di bit

            and     [si],bh     ; aggiorna pixel di sfondo

; lato destro del carattere

            pop     ax
            mov     ah,al       ; AH := bit per lato destro del carat.
            mov     al,8
            out     dx,ax       ; imposta maschera di bit

            inc     si          ; DS:SI -> lato destro del carattere
                                ;  nel buffer

            and     [si],bl     ; aggiorna pixel di primo piano

            not     ch          ; CH := maschera per lato destro del
                                ;  carattere

            xor     ah,ch       ; AH := bit per pixel di sfondo
            out     dx,ax       ; imposta maschera di bit

            and     [si],bh     ; aggiorna pixel di sfondo

; incrementa a riga successiva di pixel nel carattere

            inc     di          ; ES:DI -> byte success. nella tabella
                                ;  definizione car.
            dec     si
            add     si,Bytes
                    PerLine     ; DS:SI -> linea successiva nel buffer
                                ;  del video

            pop     cx
            loop    L20

; ripristina registri di default del controller grafico

Lexit:      mov     ax,0FF08h   ; maschera di bit di default
            out     dx,ax
```

```
                mov     ax,0005     ; registro di default di modalità
                out     dx,ax
                  .
                mov     ax,0003     ; rotaz. dati/selez. funz. di default
                out     dx,ax

                mov     ax,0F07h    ; colore ininfluente di default
                out     dx,ax

                pop     ds          ; ripristina reg. chiamante e ritorna
                pop     di
                pop     si
                mov     sp,bp
                pop     bp
                ret

_DisplayChar10ENDP

_TEXT           ENDS

                END
```

Listato 87. *Un generatore di caratteri software per le modalità grafiche EGA e VGA originali.*

Scheda InColor

La tecnica per la memorizzazione dei caratteri nel buffer del video su scheda InColor Hercules, mostrata nel Listato 88, è diversa da quella impiegata su EGA o VGA in quanto potete usare il registro di colore lettura/scrittura (1AH) della scheda InColor e la modalità di scrittura 0 per aggiornare sia i valori di pixel di primo piano sia i valori di pixel di sfondo in un'unica operazione. Di conseguenza, il processo effettivo di aggiornamento dei piani di bit si riduce ad un numero relativamente ridotto di istruzioni in linguaggio macchina.

Tuttavia, l'hardware della scheda InColor non può effettuare le operazioni di pixel AND, OR o XOR direttamente. Per fare ciò, dovete scrivere subroutine addizionali che utilizzino il registro di maschera di piano per mappare le operazioni logiche sui piani di bit (vedere Capitolo 5).

```
                TITLE   'Listato 88'
                NAME    DisplayCharInC
                PAGE    55,132

;
; Nome:         DisplayCharInC
;
; Funzione:     Visualizza un carattere nella modalità a 16 colori 720x348
;               InColor
;
; Chiamante:    Microsoft C:
```

```
;
;                    azzera DisplayCharInC(c,x,y,fgd,bkgd);
;
;                    int c;                /* codice carattere */
;
;                    int x,y;              /* pixel superiore sinistro*/
;
;                    int fgd,bkgd;         /* valori di pixel di
;                                             primo piano e di sfondo */
;

ARGc          EQU    word ptr [bp+4] ; indirizzamento di cornice di stack
ARGx          EQU    word ptr [bp+6]
ARGy          EQU    word ptr [bp+8]
ARGfgd        EQU    byte ptr [bp+10]
ARGbkgd       EQU    byte ptr [bp+12]

VARmask       EQU    word ptr [bp-2]
VAR9bits      EQU    byte ptr [bp-4]

_TEXT         SEGMENTbyte public 'CODE'
              ASSUME cs:_TEXT

              EXTRN  PixelAddrHGC:near

              PUBLIC _DisplayCharInC
_DisplayCharInC      PROC       near

              push   bp         ; mantiene registri del chiamante

              mov    bp,sp
              sub    sp,4       ; spazio di stack per variabili
                                ;  locali
              push   si
              push   di
              push   ds

; calcola indirizzo del primo pixel

              mov    ax,ARGy    ; AX := y
              mov    bx,ARGx    ; BX := x
              call   PixelAddrHGC   ; ES:BX -> buffer
                                ; CL := # bit da spostare a sinistra
                                ;  per mascherare pixel
              xor    cl,7       ; CL := # bit da ruotare a destra

              push   es         ; mantiene segmento del buffer del
                                ;  video
              mov    si,bx      ; DI := offset del buffer del video

; imposta flag per caratteri a 8 o a 9 bit

              mov    ax,40h

              mov    ds,ax      ; DS := segmento dell'area dati di
                                ;  visualizzazione del BIOS
```

```
            mov     ax,0FF00h   ; AX := maschera a 8 bit
            mov     VAR9bits,0  ; azzera questo flag

            cmp     byte ptr ds:[4Ah],90 ; CRT_COLS = 90?
            je      L01         ; salta se carat. sono larghi 8 pixel
            mov     ah,7Fh      ; AX := maschera a 9 bit
            cmp     ARGc,0C0h
            jb      L01         ; salta se codice carattere ...

            cmp     ARGc,0DFh
            ja      L01         ; ... è fuori dal range 0C0-0DFh

            inc     VAR9bits    ; imposta flag per estendere a 9 bit

L01:        ror     ax,cl       ; AX := maschera di bit in posizione
                                ;  corretta
            mov     VARmask,ax

; imposta indirizzamento tabella definizione caratteri

            mov     ax,40h
            mov     ds,ax       ; DS := segmento dell'area dati
                                ;  di visualizzazione del BIOS
            mov     ch,ds:[85h] ; CH := POINTS (righe di pixel nel
                                ;  carattere)

            xor     ax,ax
            mov     ds,ax       ; DS := zero assoluto

mov         ax,ARGc             ; AL := codice di carattere
            cmp     al,80h
            jae     L02

            mov     bx,43h*4    ; DS:BX -> vettore int 43h se car. < 80h
            jmp     short L03

L02:        mov     bx,1Fh*4    ; DS:BX -> vettore int 1Fh se car.> = 80h
            sub     al,80h      ; introduce codice carattere nella
                                ;  gamma della tabella

L03:        les     di,ds:[bx]  ; ES:DI - >inizio tabella caratteri
            mul     ch          ; AX := offset nella tabella
                                ;  definizione caratteri
                                ;  (POINTS * codice carattere)
            add     di,ax       ; DI := indirizzo definizione carat.

            pop     ds          ; DS:SI - >buffer del video

; imposta registri di controllo

            mov     dx,3B4h     ; porta di I/O del reg. di controllo

            push    cx          ; mantiene CX
            mov     ah,ARGbkgd  ; AH := valore di pixel di sfondo
            mov     cl,4
            shl     ah,cl       ; AH bit 4-7 := valore di pixel di
                                ;  sfondo
            or      ah,ARGfgd   ; AH bit 0-3 := valore di pixel di
```

```
                                ;  primo piano
            pop     cx          ; ripristina CX

            mov     al,1Ah      ; AL := numero reg colore let./scrit.
            out     dx,ax       ; imposta valore colore let./scrit.

; maschera ed imposta pixel nel buffer del video

L20:        xor     bh,bh
            mov     bl,es:[di]  ; BX := configuraz. di bit per riga
                                ;  success. di pixel
            inc     di          ; incrementa puntatore su tabella
                                ;  definizione caratteri
            cmp     VAR9bits,0

            je      L21         ; salta se carattere è largo 8 pixel

            ror     bx,1        ; copia bit meno significativo di BX
            rcl     bl,1        ;  nel bit più significativo

L21:        ror     bx,cl       ; ruota pixel in posizione

            mov     ax,5F19h    ; AH bit 6 := 1 (polarità di
                                ;  maschera)

                                ; AH bit 4-5 := 01b (modalità di
                                ;  scrittura 1)
                                ; AH bit 0-3 := 1111b (bit
                                ;  ininfluenti)
                                ; AL := 19h (reg controllo
                                ;  lettura/scrittura)
            out     dx,ax       ; imposta reg controllo let./scrit.
            or      [si],bl     ; aggiorna pixel di primo piano
            or      [si+1],bh

            mov     ah,6Fh      ; imposta modalità di scrittura 2
            out     dx,ax

            or      bx,VARmask  ; BX := configurazione di bit di
                                ;  pixel di sfondo
            or      [si],bl     ; aggiorna pixel di sfondo
            or      [si+1],bh

            add     si,2000h    ; incrementa ad area successiva di
                                ;  intercalazione
            jns     L22

            add     si,90-8000h; incrementa alla prima area di
                                ;  intercalazione


L22:        dec     ch
            jnz     L20

; ripristina valori di default dei registri InColor

            mov     ax,4019h    ; reg di controllo lettura/scrittura
```

```
                                ;  di default
            out     dx,ax

            mov     ax,071Ah    ; reg colore lettura/scrittura di
                                ;  default
            out     dx,ax

            pop     ds          ; ripristina registri e ritorna
            pop     di
            pop     si
            mov     sp,bp
            pop     bp
            ret

_DisplayCharInC     ENDP

_TEXT       ENDS

            END
```

Listato 88. *Un generatore di caratteri software per le modalità grafiche InColor Hercules.*

10
Set di caratteri alfanumerici

Tabelle di definizione caratteri
Tabelle dei caratteri alfanumerici in ROM
Tabelle dei caratteri alfanumerici in RAM

Aggiornamento della RAM del generatore di caratteri
EGA e VGA - HGC+ - Scheda InColor - MCGA

Uso dei set di caratteri basati su RAM
Set di caratteri ASCII
Set di caratteri estesi
Problemi di compatibilità con i codici di carattere estesi

Modifica della matrice di carattere visualizzato
EGA - VGA - MCGA - HGC+ e scheda InColor

Finestre grafiche in modalità alfanumeriche
HGC+ e scheda InColor - EGA e VGA - MCGA

Uno dei metodi più semplici per accelerare l'interfaccia di visualizzazione di un programma è costituito dall'uso di una modalità di visualizzazione alfanumerica. Per ottenere questo aumento di velocità, tuttavia, dovete accettare i limiti del generatore di caratteri alfanumerico del sistema di visualizzazione.

Su MDA e CGA originali, i soli caratteri visualizzabili in modalità alfanumerica erano quelli definiti in una tabella situata in ROM sull'adattatore. Il generatore di caratteri hardware di questi adattatori non era progettato per utilizzare una tabella di definizione caratteri posta in RAM. In ogni caso, l'EGA, l'MCGA, la VGA, l'HGC+ e la scheda InColor possono tutti visualizzare caratteri alfanumerici definiti in RAM.

Questo capitolo mostra come sfruttare i set di caratteri alfanumerici basati su RAM su questi sistemi di visualizzazione. Il capitolo descrive come formattare le tabelle di definizione caratteri e dove memorizzarle in RAM per poterle utilizzare nelle modalità alfanumeriche. Verranno anche messi in evidenza i pro e i contro dell'uso dei set di caratteri estesi che contengono più dei soliti 256 caratteri ASCII. Il capitolo si conclude con le tecniche per la visualizzazione di immagini grafiche in una modalità video alfanumerica.

Tabelle di definizione caratteri

Come nel caso dei generatori di caratteri software descritti nel precedente capitolo, il generatore di caratteri alfanumerici hardware di tutti i sistemi di visualizzazione IBM fa riferimento ad una tabella di definizione caratteri residente in memoria che contiene le rappresentazioni a configurazione di bit dei pixel di ogni carattere visualizzabile. A differenza delle tabelle delle modalità grafiche, la cui locazione in memoria può variare, le tabelle alfanumeriche devono trovarsi in un'area predefinita della memoria per permettere al generatore di caratteri alfanumerici di accedervi.

Tabelle dei caratteri alfanumerici in ROM

L'MDA, il CGA e gli adattatori Hercules presentano una tabella di definizione caratteri alfanumerici situata in ROM al di fuori dello spazio di indirizzo della CPU. Soltanto l'hardware del generatore di caratteri può accedervi. Il set di caratteri visualizzato da questi adattatori nelle modalità alfanumeriche non viene quindi controllato da software.

Su EGA, MCGA e VGA il generatore di caratteri alfanumerici utilizza una tabella di configurazioni di bit memorizzata in RAM piuttosto che nella ROM dedicata. Il BIOS ROM del video contiene tabelle con le quali inizializza la RAM del generatore di caratteri ogni volta che viene scelta una modalità di visualizzazione alfanumerica. Dal momento che questi sistemi di visualizzazione possono impostare modalità alfanumeriche con diverse risoluzioni verticali, le dimensioni dei caratteri alfanumerici di default varieranno di conseguenza (vedere Figura 97).

Modalità a 200 linee

Le modalità alfanumeriche a 200 linee del CGA usano una matrice di carattere 8 per 8. Nella modalità alfanumerica 80 per 25, lo schermo è quindi largo 640 pixel; nella modalità alfanumerica 40 per 25, lo schermo è largo 320 pixel. Sebbene il CGA utilizzi lo stesso set e tipo di caratteri nelle proprie modalità alfanumeriche e grafiche, le definizioni di caratteri per le modalità alfanumeriche risiedono in una ROM dedicata, a cui può accedere solo il generatore di caratteri hardware (come descritto nel Capitolo 9, le definizioni di modalità grafica si trovano nel BIOS ROM e in una tabella in RAM indirizzata dal vettore per l'interrupt 1FH).

Adattatore	Modalità video	Matrice carattere (larghezza per altezza in pixel)
MDA, HGC	Monocromatica	9 per 14
CGA	16 colori 40 per 25	8 per 8
	16 colori 80 per 25	8 per 8
EGA	16 colori 80 per 25	8 per 8 (risoluzione 200 linee)
		8 per 14 (risoluzione 350 linee)
	monocromatica 80 per 25	9 per 14
MCGA	16 colori 40 per 25	8 per 16
	16 colori 80 per 25	8 per 16
VGA	16 colori 40 per 25	8 per 8 (risoluzione 200 linee)
		8 per 14 (risoluzione 350 linee)
		9 per 16 (risoluzione 400 linee)
	16 colori 80 per 25	8 per 8 (risoluzione 200 linee)
		8 per 14 (risoluzione 350 linee)
		9 per 16 (risoluzione 400 linee)
	monocromatica 80 per 25	9 per 14 (risoluzione 350 linee)
		9 per 16 (risoluzione 400 linee)
HGC+	monocromatica 80 per 25	9 per 14
Scheda InColor	16 colori 80 per 25	9 per 14

Figura 97. *La matrice di carattere alfanumerico di default nelle varie modalità di visualizzazione.*

CONSIGLIO

Il CGA è dotato di due tabelle di caratteri 8 per 8 nella ROM del generatore di caratteri alfanumerici, un ponticello sull'adattatore seleziona quale tabella viene usata dal generatore di caratteri alfanumerici. Per default, il ponticello P3 su CGA non è collegato, quindi vengono visualizzati i soliti caratteri 8 per 8 "a doppio punto". Se collegate il ponticello P3, il generatore di caratteri alfanumerici del CGA userà il tipo di caratteri "a singolo punto" (vedere Figura 98). I caratteri "a singolo punto" risultano più ni-

tidi su alcuni monitor in quanto i loro tratti verticali sono larghi solo un pixel.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ ←—— Ponticello P3 scollegato (default)
abcdefghijklmnopqrstuvwxyz
0123456789
ABCDEFGHIJKLMNOPQRSTUVWXYZ ←—— Ponticello P3 collegato
abcdefghijklmnopqrstuvwxyz
0123456789
```

Figura 98. *Set di caratteri a doppi punti e a singoli punti su CGA.*

Modalità a 350 linee

Nelle modalità a 350 linee su MDA e sugli adattatori Hercules, i caratteri sono definiti in una matrice 8 per 14. Anche in questo caso, la tabella di definizione caratteri risiede in ROM al di fuori dello spazio di indirizzo della CPU dedicato al generatore di caratteri hardware. Dal momento che su questi adattatori la risoluzione orizzontale è di 720 pixel, ogni carattere 8 per 14 in effetti viene visualizzato in una matrice larga 9 pixel. Di conseguenza, ogni riga sullo schermo contiene 720:9, cioè 80, caratteri.

Se i caratteri vengono definiti in ROM in una matrice 8 per 14 ma visualizzati in una matrice 9 per 14, da dove proviene il pixel supplementare? Il generatore di caratteri hardware dell'MDA, delle schede Hercules, dell'EGA e della VGA (in modalità monocromatica) aggiunge un pixel supplementare alla destra di ogni riga di otto pixel di ogni carattere. Per i caratteri grafici di blocco (codici ASCII da 0C0H a 0DFH), il valore del pixel all'estrema destra viene duplicato in ogni riga. Per tutti i codici di carattere restanti, il pixel supplementare viene visualizzato con l'attributo di sfondo del carattere.

Dal momento che il nono pixel (quello all'estrema destra) dei caratteri grafici di blocco è una copia dell'ottavo pixel, questi caratteri posti uno a ridosso dell'altro possono essere utilizzati per tracciare linee orizzontali. Tutti gli altri caratteri visualizzabili sono separati uno dall'altro dal nono pixel. Il risultato della visualizzazione appare meno fitto di come apparirebbe senza lo spazio supplementare.

CONSIGLIO

Con EGA e con VGA potete controllare se il generatore di caratteri alfanumerico duplica o meno l'ottavo pixel dei caratteri grafici di blocco. Quando il bit 2 del registro di controllo modalità (10H) del controller di attributo è impostato a 1, il nono pixel è identico all'ottavo. Quando il bit 2 è impostato a 0, il nono pixel è un pixel di sfondo.

Modalità a 400 linee

Le modalità alfanumeriche di default sia dell'MCGA sia della VGA hanno una risoluzione verticale di 400 linee. I caratteri usati in queste modalità vengono definiti in una matrice 8 per 16. Su VGA, i caratteri 8 per 16 vengono visualizzati in una matrice 9 per 16, proprio come su MDA o su EGA con un monitor monocromatico.

Tabelle dei caratteri alfanumerici in RAM

L'EGA, la VGA, l'MCGA, l'HGC+ e la scheda InColor hanno tutti generatori di caratteri alfanumerici che utilizzano tabelle di definizione caratteri poste in aree predefinite di RAM. In tutti questi sistemi di visualizzazione, questa RAM si trova all'interno dello spazio di indirizzo del buffer del video. Se sapete come viene mappata la RAM del generatore di caratteri, potete scrivere dei programmi che leggono o che aggiornano le tabelle di definizione caratteri alfanumerici e quindi che modificano il set di caratteri alfanumerici visualizzato.

EGA e VGA

Nelle modalità alfanumeriche su EGA e VGA, il buffer del video è organizzato in quattro aree di memoria parallele, esattamente come nelle modalità grafiche. Nelle modalità alfanumeriche, però, solo le aree 0 e 1 contengono dati visualizzabili (vedere Figura 99). I byte con numeri pari (i codici di carattere) nello spazio di indirizzo della CPU sono posti nell'area 0 e i byte con numeri dispari (i byte di attributo) sono posti nell'area 1. Questa mappatura è invisibile alla CPU; il CRTC traduce internamente gli indirizzi dispari in offset nella mappa 1 e gli indirizzi pari in riferimenti alla mappa 0.

Il generatore di caratteri alfanumerici usa un set di tabelle a 256 caratteri memorizzate nella mappa 2. L'EGA supporta quattro di queste tabelle (vedere Figura 100); la VGA ne supporta otto (vedere Figura 101). Ogni tabella è costituita da 256 configurazioni di bit a 32 byte, quindi l'altezza massima della matrice di carattere è 32 linee di scansione. Quando la matrice di carattere visualizzato contiene meno di 32 linee, il generatore di caratteri ignora i byte eccedenti di ogni definizione carattere.

Su EGA, ognuna delle quattro tabelle di definizione caratteri alfanumerici occupa circa 16 KB. Dal momento che vengono utilizzati solo 8 KB (256 caratteri x 32 byte per carattere), 8 KB di RAM inutilizzata si trovano in coda ad ogni tabella. Su VGA, queste aree inutilizzate nella mappa 2 possono contenere ulteriori definizioni di caratteri. Naturalmente, nella scrittura di un'applicazione che deve operare sia su EGA sia su VGA, dovreste evitare di utilizzare queste tabelle aggiuntive in quanto l'EGA non le supporta.

CONSIGLIO

Su EGA IBM, che potrebbe essere dotato di meno di 256 KB di RAM video, il numero di tabelle di definizione caratteri che potete caricare nella RAM del video dipende dalla quantità di RAM installata sulla scheda. Ad esempio, senza la scheda di espansione di memoria grafica dell'IBM, una scheda EGA IBM

ha solo 64 KB di RAM video, quindi ogni mappa di memoria video nelle modalità alfanumeriche contiene solo 16 KByte, e solo una tabella di definizione caratteri può essere contenuta nella mappa 2.

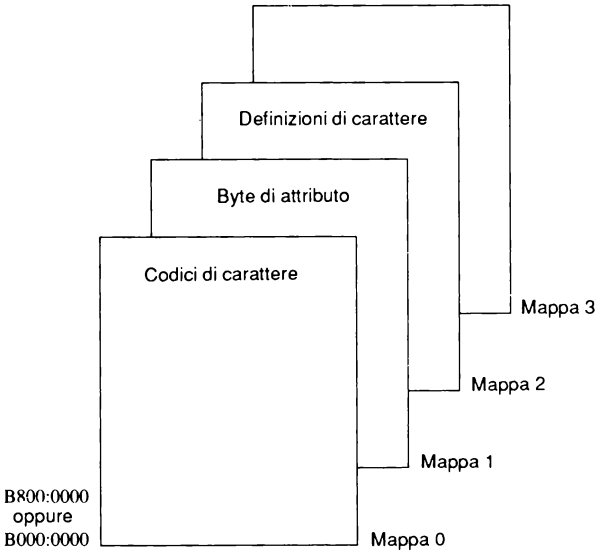


Figura 99. *Disposizione della RAM del video nelle modalità alfanumeriche EGA e VGA*

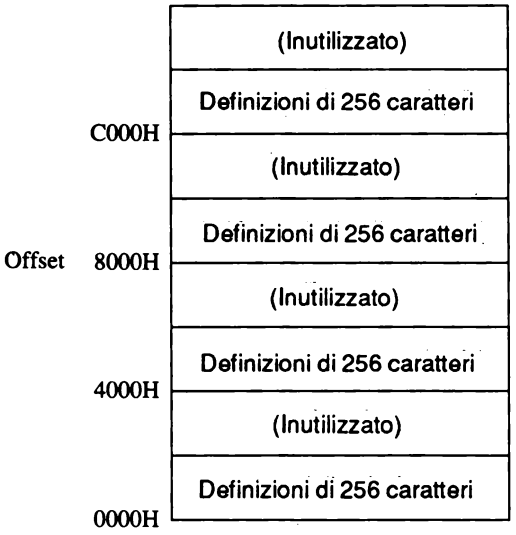


Figura 100. *RAM del generatore di caratteri nell area 2 di memoria video EGA*

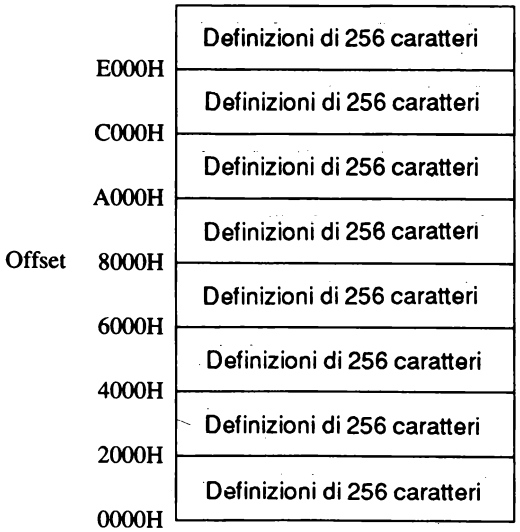


Figura 101. *RAM del generatore di caratteri nell area 2 di memoria del VGA.*

HGC+

La RAM del generatore di caratteri su HGC+ inizia a B000:4000 e si estende fino alla fine della RAM video disponibile a B000:FFFF (vedere Figura 102). Potete riempire questa intera area di 48 KB con definizioni di caratteri. Ogni definizione di carattere è lunga 16 byte, quindi una tabella che definisce 256 caratteri occupa 4 KB. Di conseguenza, questa RAM può contenere 3072 definizioni di carattere.

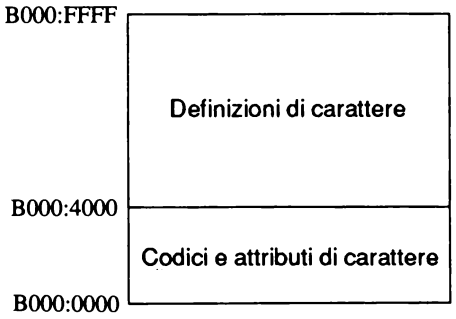


Figura 102. *Disposizione della RAM del video nelle modalità alfanumeriche su HGC+.*

CONSIGLIO

Se l'HGC+ è configurata in modo che la RAM video al di sopra di B000:8000 venga mascherata dallo spazio di indirizzo della CPU (cioè, se il bit 1 dello switch di configurazione a 3BFH è impostato a 0), allora solo i 16 KB di RAM tra B000:4000 e B000:7FFF possono essere utilizzati per le definizioni di caratteri.

Scheda InColor

Su scheda InColor la RAM del generatore di caratteri occupa la stessa gamma di indirizzi occupata su HGC+, cioè da B000:4000 a B000:FFFF. Inoltre, ogni definizione di carattere InColor è lunga 16 byte. A differenza dell'HGC+, però, la modalità a 16 colori della scheda InColor usa tutti i quattro piani di bit di questa gamma di indirizzi per le definizioni di carattere (vedere Figura 103).

Grazie a questo fatto, potete controllare il valore di ogni pixel in ogni carattere da voi definito. Potete anche programmare la scheda InColor in modo che diversi piani di bit definiscano diversi caratteri; quando i caratteri vengono visualizzati, i loro byte di attributo selezionano i piani di bit da utilizzare. Caricando ognuno dei quattro piani di bit con diverse definizioni di carattere, potete mantenere in RAM fino a 12.288 (3072 x 4) definizioni di carattere. In alternativa, per mantenere la compatibilità con l'HGC+, potete caricare tutti i quattro piani di bit con le stesse configurazioni di bit.

CONSIGLIO

Usando sia EGA sia schede Hercules, prestate attenzione nel passaggio da una modalità alfanumerica che utilizza una tabella di definizione caratteri basata su RAM ad una modalità grafica. La stessa RAM che contiene i dati di pixel nelle modalità grafiche viene usata per memorizzare le definizioni di carattere nelle modalità alfanumeriche. Potreste compromettere o cancellare le tabelle di definizione carattere aggiornando il buffer del video in una modalità grafica e quindi tornando ad una modalità alfanumerica.

MCGA

A differenza dell'EGA e della VGA, l'MCGA non ha mappe parallele di memoria nelle quali memorizzare le definizioni di carattere. Le definizioni di caratteri alfanumerici vengono invece mantenute nei 32 KB di RAM video tra A000:0000 e A000:7FFF. Potete memorizzare fino a quattro tabelle di definizione caratteri da 8 KB a A000:0000, A000:2000, A000:4000 e A000:6000 (vedere Figura 104).

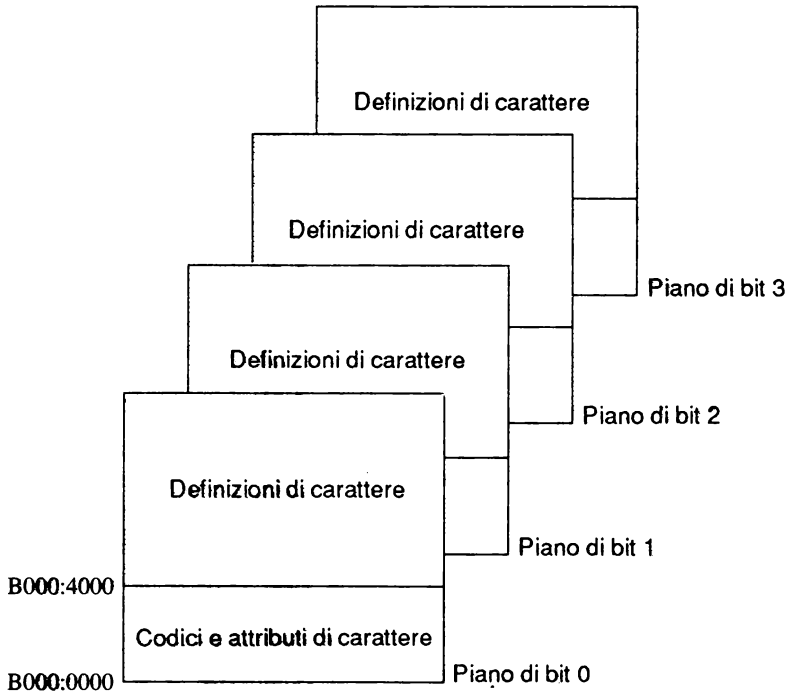


Figura 103. *Disposizione della RAM video nelle modalità alfanumeriche su scheda InColor Hercules. Le definizioni di carattere iniziano a B000:4000 in tutti i quattro piani di bit.*

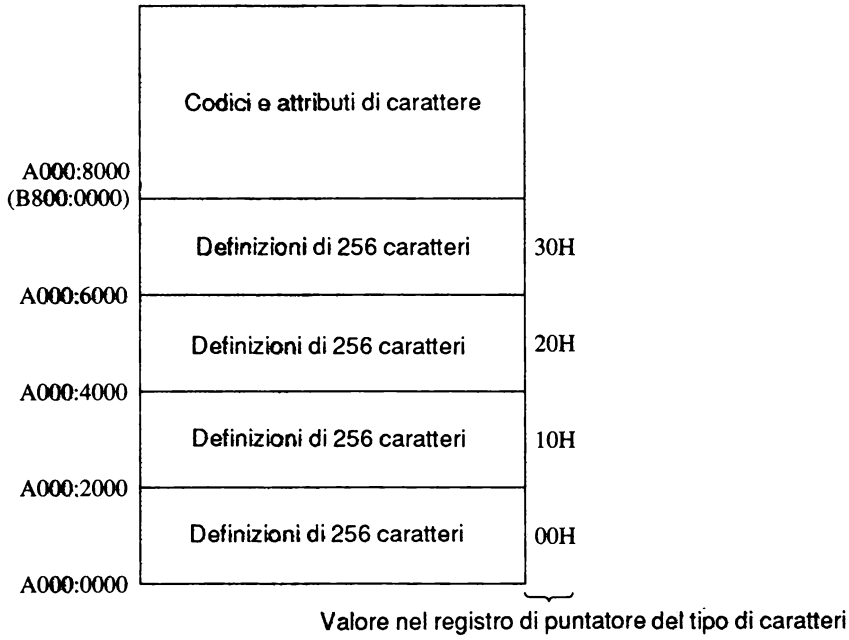


Figura 104. *Disposizione della RAM video nelle modalità alfanumeriche MCGA.*

Il formato delle tabelle di definizione carattere dell'MCGA è molto diverso da quello delle altre tabelle trattate finora. Ogni tabella da 8 KB è divisa in 16 elenchi da 512 byte di codici di caratteri e di configurazioni di bit (vedere Figura 105). Ogni elenco corrisponde ad una linea di scansione dei caratteri definiti; il primo elenco rappresenta le configurazioni di bit della linea di scansione superiore di ogni carattere, il secondo elenco corrisponde alla seconda linea di scansione e così via (vedere Figura 106). Dal momento che ci sono 16 elenchi, l'altezza massima di un carattere è 16 linee.

```
            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
A000:0400  00 00 01 7E 02 7E 03 00 04 00 05 00 06 00 07 00  ...~.~..........
A000:0410  08 FF 09 00 0A FF 0B 1E 0C 3C 0D 3F 0E 7F 0F 00  .........<.?....
A000:0420  10 C0 11 06 12 18 13 66 14 7F 15 C6 16 00 17 18  .......f........
A000:0430  18 18 19 18 1A 00 1B 00 1C 00 1D 00 1E 00 1F 00  ................
A000:0440  20 00 21 18 22 66 23 00 24 7C 25 00 26 38 27 30   .!."f#.$|%.&8'0
A000:0450  28 0C 29 30 2A 00 2B 00 2C 00 2D 00 2E 00 2F 00  (.)0*.+.,.-.../.
A000:0460  30 7C 31 18 32 7C 33 7C 34 0C 35 FE 36 38 37 FE  0|1.2|3|4.5.687.
A000:0470  38 7C 39 7C 3A 00 3B 00 3C 00 3D 00 3E 00 3F 7C  8|9|:.;.<.=.>.?|
A000:0480  40 00 41 10 42 FC 43 3C 44 F8 45 FE 46 FE 47 3C  @.A.B.C<D.E.F.G<
A000:0490  48 C6 49 3C 4A 1E 4B E6 4C F0 4D C6 4E C6 4F 38  H.I<J.K.L.M.N.O8
A000:04A0  50 FC 51 7C 52 FC 53 7C 54 7E 55 C6 56 C6 57 C6  P.Q|R.S|T~U.V.W.
A000:04B0  58 C6 59 66 5A FE 5B 3C 5C 00 5D 3C 5E 6C 5F 00  X.YfZ.[<\.]<^l_.
A000:04C0  60 18 61 00 62 E0 63 00 64 1C 65 00 66 38 67 00  `.a.b.c.d.e.f8g.
A000:04D0  68 E0 69 18 6A 06 6B E0 6C 38 6D 00 6E 00 6F 00  h.i.j.k.l8m.n.o.
A000:04E0  70 00 71 00 72 00 73 00 74 10 75 00 76 00 77 00  p.q.r.s.t.u.v.w.
A000:04F0  78 00 79 00 7A 00 7B 0E 7C 18 7D 70 7E 76 7F 00  x.y.z.{.|.}p~v..
A000:0500  80 3C 81 CC 82 18 83 38 84 CC 85 30 86 6C 87 00  .<.....8...0.l..
A000:0510  88 38 89 CC 8A 30 8B 66 8C 3C 8D 30 8E C6 8F 38  .8...0.f.<.0...8
A000:0520  90 60 91 00 92 3E 93 38 94 C6 95 30 96 78 97 30  .`...>.8...0.x.0
A000:0530  98 C6 99 C6 9A C6 9B 18 9C 6C 9D 66 9E CC 9F 1B  .........l.f....
A000:0540  A0 30 A1 18 A2 30 A3 30 A4 76 A5 00 A6 6C A7 6C  .0...0.0.v...l.l
A000:0550  A8 30 A9 00 AA 00 AB C0 AC C0 AD 18 AE 00 AF 00  .0..............
```

Figura 105. *Uno dei 16 elenchi di codici di carattere e di configurazioni di bit nella RAM del generatore di caratteri MCGA. Questa tabella definisce le configurazioni di bit per la terza linea di scansione di ogni carattere. I codici di carattere si trovano nei byte con numeri pari. I byte con numeri dispari contengono le corrispondenti configurazioni di bit.*

```
            0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
A000:0000  00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00  ................
A000:0010  08 FF 09 00 0A FF 0B 00 0C 00 0D 00 0E 00 0F 00  ................

A000:0200  00 00 01 00 02 00 03 00 04 00 05 00 06 00 07 00  ................
A000:0210  08 FF 09 00 0A FF 0B 00 0C 00 0D 00 0E 00 0F 00  ................

A000:0400  00 00 01 7E 02 7E 03 00 04 00 05 00 06 00 07 00  ...~.~..........
A000:0410  08 FF 09 00 0A FF 0B 1E 0C 3C 0D 3F 0E 7F 0F 00  .........<.?....

A000:0600  00 00 01 81 02 FF 03 00 04 00 05 18 06 18 07 00  ................
A000:0610  08 FF 09 00 0A FF 0B 0E 0C 66 0D 33 0E 63 0F 18  .........f.3.c..

A000:0800  00 00 01 A5 02 DB 03 6C 04 10 05 3C 06 3C 07 00  .......l...<.<..
A000:0810  08 FF 09 00 0A FF 0B 1A 0C 66 0D 3F 0E 7F 0F 18  .........f.?....

A000:0A00  00 00 01 81 02 FF 03 FE 04 38 05 3C 06 7E 07 00  .........8.<.~..
A000:0A10  08 FF 09 3C 0A C3 0B 32 0C 66 0D 30 0E 63 0F DB  ...<...2.f.0.c..

A000:0C00  00 00 01 81 02 FF 03 FE 04 7C 05 E7 06 FF 07 18  .........|......
A000:0C10  08 E7 09 66 0A 99 0B 78 0C 66 0D 30 0E 63 0F 3C  ...f...x.f.0.c.<

A000:0E00  00 00 01 BD 02 C3 03 FE 04 FE 05 E7 06 FF 07 3C  ...............<
A000:0E10  08 C3 09 42 0A BD 0B CC 0C 3C 0D 30 0E 63 0F E7  ...B.....<.0.c..

A000:1000  00 00 01 99 02 E7 03 FE 04 7C 05 E7 06 7E 07 3C  .........|...~.<
A000:1010  08 C3 09 42 0A BD 0B CC 0C 18 0D 30 0E 63 0F 3C  ...B.......0.c.<

A000:1200  00 00 01 81 02 FF 03 7C 04 38 05 18 06 18 07 18  .......|.8......
A000:1210  08 E7 09 66 0A 99 0B CC 0C 7E 0D 70 0E 67 0F DB  ...f.....~.p.g..

A000:1400  00 00 01 81 02 FF 03 38 04 10 05 18 06 18 07 00  .......8........
A000:1410  08 FF 09 3C 0A C3 0B CC 0C 18 0D F0 0E E7 0F 18  ...<............

A000:1600  00 00 01 7E 02 7E 03 10 04 00 05 3C 06 3C 07 00  ...~.~.....<.<..
A000:1610  08 FF 09 00 0A FF 0B 78 0C 18 0D E0 0E E6 0F 18  .......x........
```

Figura 106. *Le definizioni di carattere dell'MCGA per le prime 12 linee di scansione dei primi 16 caratteri. La linea di scansione superiore per ogni carattere viene definita a partire da A000:0000, la seconda linea di scansione a partire da A000:2000, e così via (sono mostrati solo i primi 32 byte di ogni elenco di 512 byte).*

Aggiornamento della RAM del generatore di caratteri

Dopo aver creato una tabella di definizioni di carattere (trattata nel Capitolo 9), dovete rendere la tabella accessibile al generatore di caratteri hardware posizionandola correttamente all'interno del buffer del video. Un metodo per effettuare questa operazione è quello di creare la tabella in RAM (al di fuori del buffer del video) e quindi di copiarla sulla RAM del generatore di caratteri. Potete anche trasferire direttamente la tabella da un file su disco nella RAM del generatore di caratteri. Entrambe le tecniche funzionano su qualsiasi sistema di visualizzazione trattato in questo libro.

EGA e VGA

Per copiare una tabella di definizione caratteri nella mappa 2 di memoria video, dovete programmare sia il registro di modalità memoria del sequencer sia il relativo registro di maschera di mappa, oltre ai registri di modalità e ai registri vari del controller grafico, per rendere direttamente indirizzabile la mappa 2 della memoria. Potete quindi copiare le definizioni di carattere su una qualsiasi locazione di tabella disponibile nella mappa 2. Dopo aver aggiornato la mappa 2, riportate i registri del sequencer e del controller grafico ai valori appropriati per la modalità alfanumerica che state utilizzando.

Il Listato 89 mostra come vengono programmati il sequencer ed il controller grafico sia su EGA sia su VGA per rendere accessibile la RAM del generatore di caratteri della mappa 2. Il Listato 90 è la routine contraria, che riporta i registri del sequencer e del controller grafico ai valori di default della modalità alfanumerica. Potete utilizzare le routine dei Listati 89 e 90 in un programma che copia le definizioni di carattere direttamente da un file sulla RAM del generatore di caratteri (come mostrato nei Listato 91 e 92).

```
                TITLE   'Listato 89'
                NAME    CGenModeSet
                PAGE    55,132

;
; Nome:         CGenModeSet
;
;               Accesso diretto alla RAM del generatore di caratteri
;               alfanumerici EGA e VGA
;
; Chiamante:    Microsoft C:
;
;                       azzera CGenModeSet();
;
```

```
DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                PUBLIC _CGenModeSet
_CGenModeSet    PROC    near

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                push    si

; programma il sequencer

                cli                 ; disabilita interrupt
                mov     dx,3C4h     ; indirizzo di porta del sequencer
                mov     si,offset DGROUP:SeqParms
                mov     cx,4

L01:            lodsw               ; AH := valore del registro del sequencer
                                    ; AL := numero registro
                out     dx,ax       ; programma il registro
                loop    L01
                sti                 ; abilita interrupt

; programma il controller grafico

                mov     dl,0CEh     ; DX := 3CEH (indirizzo di porta del
                                    ;       controller grafico)
                mov     si,offset DGROUP:GCParms
                mov     cx,3

L02:            lodsw               ; programma il controller grafico
                out     dx,ax
                loop    L02

                pop     si
                pop     bp
                ret

_CGenModeSet    ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

; Il formato dei parametri è:  byte meno significativo:  numero registro
;                              byte più significativo:   valore del reg

SeqParms        DW      0100h       ; ripristino sincrono
                DW      0402h       ; CPU scrive solo su mappa 2
                DW      0704h       ; indirizzamento sequenziale
                DW      0300h       ; azzera ripristino sincrono
```

```
GCParms         DW      0204h       ; seleziona mappa 2 per letture di CPU
                DW      0005h       ; disabilita indirizzamento dispari-pari
                DW      0006h       ; mappa inizia a A000:0000

_DATA           ENDS

                END
```

Listato 89. *Uso della RAM del generatore di caratteri su EGA e VGA.*

```
                TITLE   'Listato 90'
                NAME    CGenModeClear
                PAGE    55,132

;
; Nome:         CGenModeClear
;
;               Ripristina modalità alfanumerica EGA o VGA dopo l'accesso
;               alla RAM del generatore di caratteri
;
; Chiamante:    Microsoft C:
;
;                       azzera CGenModeClear();
;


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                PUBLIC _CGenModeClear
_CGenModeClear PROC     near

                push    bp          ; mantiene registri chiamante
                mov     bp,sp
                push    si

; programma il sequencer

                cli                 ; disabilita interrupt
                mov     dx,3C4h     ; indirizzo di porta del sequencer
                mov     si,offset DGROUP:SeqParms
                mov     cx,4

L01:            lodsw               ; AH := valore del registro del sequencer
                                    ; AL := numero registro
                out     dx,ax       ; programma il registero
                loop    L01
                sti                 ; abilita interrupt
```

```
; programma controller grafico

                mov     dl,0CEh     ; DX := 3CEH (indirizzo di porta del
                                    ;             controller grafico)
                mov     si,offset DGROUP:GCParms
                mov     cx,3

L02:            lodsw               ; programma controller grafico
                out     dx,ax
                loop    L02

                mov     ah,0Fh      ; AH := numero funzione INT 10H
                int     10h         ; attiva modalità video

                cmp     al,7
                jne     L03         ; salta se non è modalità monocromatica

                mov     ax,0806h    ; programma controller grafico
                out     dx,ax       ;  per iniziare mappa a B000:0000

L03:            pop     si
                pop     bp
                ret

_CGenModeClear ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

; Il formato dei parametri è:  byte meno significativo:  numero registro
;                              byte più significativo:   valore del reg

SeqParms        DW      0100h       ; ripristino sincrono
                DW      0302h       ; CPU scrive su mappe 0 e 1
                DW      0304h       ; indirizzamento dispari-pari
                DW      0300h       ; azzera ripristino sincrono

GCParms         DW      0004h       ; seleziona mappa 0 per letture di CPU
                DW      1005h       ; abilita indirizzamento dispari-pari
                DW      0E06h       ; mappa inizia a B800:0000

_DATA           ENDS

                END
```

Listato 90. *Ripristino della RAM del generatore di caratteri su EGA e VGA.*

```
                TITLE  'Listato 91'
                NAME   CGenRead1
                PAGE   55,132

;
; Nome:         CGenRead1
;
;               Legge 256 definizioni di caratteri nella RAM dei caratteri
;               EGA e VGA
;
; Chiamante:    Microsoft C:
;
;                       azzera CGenRead1(f,p);
;
;                       int        f;  /* gestione file */
;                       int        p;  /* byte per definizione carattere */
;

ARGf            EQU     [bp+4]
ARGp            EQU     [bp+6]

CGenRAMSeg      EQU     0A000h          ; inizio della RAM del
                                        ;  generatore di caratteri
CGenRAMOffset   EQU     0
CGenDefSize     EQU     32              ; dimensione in byte di una
                                        ;  definiz. di car.

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC _CGenRead1
_CGenRead1      PROC   near

                push   bp               ; mantiene registri
                mov    bp,sp
                push   ds
                push   si
                push   di

; azzera RAM definizione caratteri

                mov    di,CGenRARSeg
                mov    es,di            ; ES := segmento RAM del
                                        ;  generatore caratteri
                mov    di,CGenRAMOffset

                mov    cx,256*CGenDefSize/2 ; CX := numero di word a zero
                xor    ax,ax
                rep    stosw

; carica definizioni caratteri da file

                mov    cx,256           ; assume 256 definizioni
                                        ;  caratteri nel file
```

```
                mov     bx,ARGf         ; BX := gestione file
                mov     si,ARGp         ; CX := byte per definizione
                                        ;  carattere
                push    es
                pop     ds
                mov     dx,CGenRAMOffset      ; DS:DX -> inizio della RAM del
                                              ;  generatore di caratteri

L01:            xchg    cx,si           ; CX := numero di byte da leggere
                                        ; SI := contatore di loop
                mov     ah,3Fh          ; AH := numero di funzione INT 21H
                int     21h
                add     dx,CGenDefSize  ; DS:DX -> successiva definizione
                                        ;  car. in RAM
                xchg    cx,si           ; CX := contatore di loop
                                        ; SI := numero di byte da leggere
                loop L01

                pop     di              ; ripristina registri ed esce
                pop     si
                pop     ds
                pop     bp
                ret

_CGenRead1      ENDP

_TEXT           ENDS

                END
```

Listato 91. *Caricamento di definizioni di carattere su EGA o VGA.*

```
/* Listato 92 */

#include        <fontl.h>
#include        <stdio.h>

main(argc,argv)
int             argc;
char            **argv;
{
                int     i;
                int     FileHandle;
                int     Points;     /* byte per definizione carattere */
                long    lseek();
                long    FileSize;

                if (argc !=2)       /* verifica nome file */
                {
                  printf ( "\nNessun nome file specificato\n" );

                  exit ( 1 );
                }

                FileHandle = open ( argv[1], O_RDONLY );/* apre file */
```

```
            if ( FileHandle == -1 )
            {
              printf ( "\n'%s' non apribile\n", argv[1] );
              exit ( 2 );
            }

            CGenModeSet();     /* rende indirizzabile RAM generat. car. */

            FileSize = lseek( FileHandle, 0L, SEEK_END );
                               /* rileva dimens. file */
            Points = FileSize / 256;     /* determina dimensione */
                                         /*    caratteri */

            lseek( FileHandle, 0L, SEEK_SET ); /* inizio del file */

            CGenRead1( FileHandle, Points );

            CGenModeClear();   /* ripristina prec. modalità */
                               /*    alfanumerica */
}
```

Listato 92. *Chiamata di CGenRead1 da un programma in C.*

Un metodo più rapido e più trasportabile per caricare definizioni di caratteri in RAM è quello di usare la funzione 11H di INT 10H con AL = 0 (vedere Listati 93 e 94). Quando utilizzate la funzione INT 10H, potete aggiornare selettivamente una qualsiasi area di una tabella nella mappa 2 scegliendo i valori appropriati per DX (l'offset di carattere nella tabella) e per CX (il numero di definizioni di carattere da aggiornare). Per usare questa funzione del BIOS del video, dovete per prima cosa memorizzare la tabella di definizione caratteri in un buffer intermedio. Questa tecnica consuma più memoria della lettura delle definizioni di caratteri direttamente da disco, ma produce un codice più veloce.

```
            TITLE   'Listato 93'
            NAME    CGenRead2
            PAGE    55,132

;
; Nome:     CGenRead2
;
;           Usa il BIOS per leggere 256 definizioni di caratteri
;           nella RAM dei caratteri EGA e VGA
;
; Chiamante: Microsoft C:
;
;                  azzera CGenRead2(f);
;
;                  int        f;  /* gestione file */

ARGf        EQU     [bp+4]

DGROUP      GROUP   _DATA
```

```
_TEXT          SEGMENTbyte public 'CODE'
               ASSUME cs:_TEXT

               PUBLIC _CGenRead2
_CGenRead2     PROC   near

               push   bp              ; mantiene registri
               mov    bp,sp

; carica definizioni caratteri da file

               mov    cx,256*32       ; assume 256 definizioni car. da
                                      ;  32 byte nel file
               mov    bx,ARGf         ; BX := gestione file
               mov    dx,offset DGROUP:CharBuf
                                      ; DS:DX - >inizio del buffer

               mov    ah,3Fh          ; AH := numero di funzione INT 21H
               int    21h             ; legge file
                                      ; AX := numero di byte letti

; richiama BIOS del video per caricare RAM del generatore di caratteri

               pus    ds
               pop    es
               mov    bp,offset DGROUP:CharBuf
                                      ; ES:BP - >definizioni caratteri
               mov    bl,0            ; BL := blocco RAM del generatore
                                      ;  di car. da caricare
               mov    bh,ah           ; AH := byte per carattere
                                      ; (numero di byte letti)/256
               mov    cx,256          ; numero di definizioni carattere
                                      ;  da memorizzare
               xor    dx,dx           ; primo numero di carattere
               mov    ax,1100h        ; AH := 11H (num. funz. INT 10H)
                                      ; AL := 0 (numero sottofunzione)
               int    10h

               pop    bp              ; ripristina BP ed esce
               ret

_CGenRead2     ENDP

_TEXT          ENDS

_DATA          SEGMENTword public 'DATA'

CharBuf        DB     256*32 dup(?)

_DATA          ENDS

               END
```

Listato 93. *Uso del BIOS per caricare definizioni di carattere.*

```
/* Listato 94 */

#include      <fcntl..h>

main(argc,argv)
int           argc;
char          **argv;
{
              int    i;
              int    FileHandle;

              if (argc !=2)                    /* verifica nome file */
              {
                printf( "\nNessun nome file specificato\n" );
                exit( 1 );
              }

              FileHandle=open(argv[1],O_RDONLY); /* apre file */

              if ( FileHandle == -1 )
              {
                printf( "\n'%s' non apribile\n", argv[1] );
                exit( 2 );
              }

              CGenRead2( FileHandle );     /* richiama BIOS video per
                                              caricare file */
                                           /*  nella RAM del generatore
                                              caratteri */
}
```

Listato 94. *Chiamata di CGenRead2 da un programma in C.*

I servizi della funzione 11H di INT 10H possono anche aggiornare la RAM del generatore di caratteri dalle tabelle di carattere nel BIOS ROM. Per utilizzare una delle tabelle di definizione caratteri del BIOS ROM, richiamate la funzione 11H di INT 10H con AL = 1 (per le definizioni dei caratteri 8 per 14) o AL = 2 (per le definizioni 8 per 8) (vedere Listato 95).

```
              mov    ax,1102h    ; AH := numero funzione INT 10H
                                 ; AL := 02h (carica caratteri 8x8 BIOS ROM)
              mov    bl,0        ; BL := banco di RAM del generatore di car.
              int    10h         ; carica set di caratteri alfanumerici
```

Listato 95. *Uso di una tabella di definizioni caratteri BIOS ROM.*

HGC+

Lo spostamento di una tabella di definizione caratteri in RAM è più semplice su HGC+ in quanto l'indirizzamento della memoria è più facile. La RAM del generatore di caratteri è mappata linearmente, a partire da B000:4000. Dal momento che ogni tabella di 256 caratteri occupa 4 KB (256x16), le successive tabelle da 256 caratteri partono da B000:5000, B000:6000, e così via.

Dal momento che la memoria dell'HGC+ non presenta piani di bit, potete accedere alla RAM del generatore di caratteri semplicemente come avviene con qualsiasi altra RAM di sistema. Potete, ad esempio, usare una singola istruzione *REP MOVSB* per spostare configurazioni di bit nella RAM del generatore di caratteri da una qualsiasi altra parte della RAM di sistema, o potete leggere una tabella di definizione caratteri direttamente in RAM a partire da un file di disco. Ad esempio, potete modificare il Listato 91 per leggere un file direttamente nella RAM del generatore di caratteri dell'HGC+ modificando i valori di *CGenRAMSeg* in B000H, *CGenStartOffset* in 4000H e *CGenDefSize* in 16.

Scheda InColor

Sebbene la scheda InColor utilizzi tutti i quattro piani di bit per memorizzare le definizioni di caratteri, potete usare virtualmente la stessa routine impiegata su HGC+ per copiare le configurazioni di bit nella RAM di generatore di caratteri. L'unica differenza è rappresentata dal fatto che dovete selezionare quale dei quattro piani di bit deve essere aggiornato. Questo viene realizzato impostando i bit da 4 a 7 del registro di maschera di piano (18H) per proteggere in scrittura uno o più piani di bit. Per mantenere la compatibilità con l'HGC+, impostate questi quattro bit a 0 in modo che tutti i quattro piani di bit contengano le stesse configurazioni di bit.

MCGA

Come nel caso degli adattatori Hercules, la RAM del generatore di caratteri della MCGA é mappata linearmente nel buffer del video. Di conseguenza, potete aggiornare le definizioni di carattere dell'MCGA semplicemente scrivendo le configurazioni di bit nel formato appropriato nelle tabelle di definizione caratteri.

Se però aggiornate direttamente le tabelle di definizione caratteri dell'MCGA, il vostro programma deve memorizzare le configurazioni di bit ed i codici di carattere nel formato previsto dal generatore di caratteri dell'MCGA. Di solito è consigliabile utilizzare la funzione 11H di INT 10H per copiare le definizioni di carattere nella RAM del generatore di caratteri dell'MCGA. Questa funzione del BIOS del video traduce le tabelle di definizione caratteri dal formato lineare utilizzato su EGA e VGA negli elenchi formattati utilizzati su MCGA.

L'MCGA è diversa dagli altri sistemi di visualizzazione trattati in questo libro in quanto il suo generatore di caratteri alfanumerici non preleva le configurazioni di bit dalle tabelle a A000:0000 mentre genera i caratteri. Al contrario, il generatore di caratteri usa due tabelle interne di definizione caratteri, chiamate pagine di tipi di caratteri. Per visualizzare i caratteri di una delle quattro tabelle della RAM del video, dovete caricare la tabella in una delle pagine di tipi di caratteri del generatore di caratteri. Il Listato 96 mostra come fare.

```
                TITLE   'Listato 96'
                NAME    SetFontPages
                PAGE    55,132

;
; Nome:         SetFontPages
;
;               Aggiorna pagine di tipi di caratteri MCGA
;
; Chiamante:    Microsoft C:
;
;                       azzera SetFontPages(n0,n1);
;
;                       int         n0,n1;
                                        /* valori di pagine di tipi car. */
;

ARGn0           EQU     [bp+4]
ARGn1           EQU     [bp+6]

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC _SetFontPages
_SetFontPages   PROC    near

                push    bp              ; mantiene registri del chiamante
                mov     bp,sp

                mov     ax,1103h        ; AH := numero funzione INT 10H
                                        ; AL := 3 (imposta specif.di blocco)
                mov     bl,ARGn1        ; BL := valore dei bit 2-3
                shl     bl,1
                shl     bl,1            ; BL bit 2-3 := n1
                or      bl,ARGn0        ; BL bit 0-1 := n0
                int     10h             ; carica pagine di tipi di carattere

                pop     bp
                ret

_SetFontPages   ENDP

_TEXT           ENDS
                END
```

Listato 96. *Caricamento delle pagine di tipi di carattere su una MCGA.*

Di conseguenza, la visualizzazione di un nuovo set di caratteri alfanumerici su MCGA è un processo a due fasi. Per prima cosa, memorizzate le tabelle di definizione carattere in uno o più dei quattro blocchi di 8 KB di RAM video riservati a questo scopo. Successivamente aggiornate le pagine di tipi di carattere del generatore di caratteri per visualizzare i caratteri.

Uso dei set di caratteri basati su RAM

Quando utilizzate i caratteri definiti in una tabella basata su RAM, dovete scegliere come il generatore di caratteri alfanumerici dovrà decodificare i codici e gli attributi di carattere memorizzati nell'area visualizzata del buffer del video. L'uso del solito set di 256 caratteri ASCII, con codici di carattere a 8 bit ed attributi di carattere a 8 bit, è il metodo più semplice. Tuttavia, per visualizzare contemporaneamente più di 256 caratteri diversi o per passare rapidamente da un set di caratteri all'altro, dovete usare una gamma più ampia di codici di caratteri "estesi" ed un diverso set di attributi.

Set di caratteri ASCII

Il metodo più semplice per personalizzare i caratteri alfanumerici è quello di utilizzare codici ASCII e attributi a 8 bit con una tabella di definizione caratteri basata su RAM. Dal momento che esistono solo 256 codici di carattere ASCII, potete visualizzare un solo set di 256 caratteri per volta. In ogni caso, i codici di carattere ed i byte di attributo memorizzati nell'area visualizzata del buffer del video mantengono il loro solito formato, quindi il software che non sa nulla delle definizioni di carattere basate su RAM può operare senza alcuna modifica mentre visualizza il set di caratteri basato su RAM.

EGA, VGA e MCGA

Ogni volta che selezionate una modalità video alfanumerica utilizzando il BIOS del video, il generatore di caratteri alfanumerici viene configurato per visualizzare i caratteri definiti nella prima tabella della RAM del generatore di caratteri. Di conseguenza, per visualizzare un diverso set di caratteri ASCII, è sufficiente aggiornare la tabella. Come descritto prima, la funzione 11H di INT 10H fornisce un comodo meccanismo per effettuare questa operazione. Questa stessa funzione del BIOS permette anche di visualizzare i 256 caratteri definiti in una qualsiasi delle altre tabelle di definizione caratteri, come descritto più avanti nel corso di questo capitolo.

HGC+ e scheda InColor

Quando avviate una HGC+ o una scheda InColor, il generatore di caratteri alfanumerici utilizza per default la tabella di definizione caratteri basata su ROM. Per visualizzare un diverso set di caratteri ASCII, configurate il generatore di caratteri alfanumerici in modo che utilizzi la tabella basata su RAM (vedere Listato 97), quindi caricate una tabella di definizione caratteri nella RAM del video a B000:4000.

Per fare ciò, impostate a 1 il bit 0 del registro di Modalità-x dell'adattatore (14H). Ciò fa sì che l'adattatore visualizzi i caratteri definiti nella tabella in RAM a B000:4000. Inoltre, impostate a 1 il bit 0 del registro di commutazione configurazione (3BFH) per rendere indirizzabile a B000:4000 la RAM del generatore di caratteri. Dopo aver aggiornato la RAM del generatore di caratteri, potrete proteggerla da successive modifiche reimpostando il bit 0 del registro di commutazione configurazione.

```
mov     dx,3B4h
mov     ax,0114h    ; AH bit 0 := 1 (abilita gener. di car. RAM)
                    ;
                    ; AL := 14h (numero registro Modalità-x)
out     dx,ax
mov     dl,0BFh     ; DX := 3BFh (registro di commut. configur.)
mov     al,1        ; AL bit 0 := 1 (rende la RAM indirizzabile
                    ;  a B000:4000)
out     dx,al


                    ; (aggiorna RAM del generatore di caratteri)


mov     dx,3BFh     ; DX := 3BFh (reg. di commut. di configur.)
mov     al,0        ; AL bit 0 := 0 (esclude RAM a B000:4000 da
                    ;  mappa di memoria)
out     dx,al
```

Listato 97. *Configurazione di un'HGC+ o di una scheda InColor per l'aggiornamento della RAM del generatore di caratteri.*

L'aggiornamento della RAM del generatore di caratteri risulta più complicato su scheda InColor in quanto tutti i quattro piani di bit vengono utilizzati per le definizioni di carattere. La complessità risiede nel modo in cui vengono visualizzati i colori per i caratteri definiti nei piani di bit. Il colore di un carattere non viene determinato solo dai propri attributi di primo piano e di sfondo, ma anche dai piani di bit utilizzati per definire la relativa configurazione di pixel.

La scheda InColor unisce i valori di pixel della definizione di un carattere (nella RAM del generatore di carattere) agli attributi di primo piano e di sfondo del carattere (nell'area visualizzata del buffer del video) per produrre un attributo a 4 bit per ogni pixel del carattere. La logica utilizzata è:

```
(valore_pixel AND attributo_primopiano) OR

(NOT valore_pixel AND attributo_sfondo)
```

Nell'esempio della Figura 107, uno dei pixel di un carattere ha il valore 2 (0010B) nella tabella di definizione carattere. Il byte di attributo del carattere nel buffer del video

specifica un valore di primo piano di 0 ed un valore di sfondo di 7 (0111B). La scheda InColor visualizza quindi questo pixel con un attributo di (2 AND 0) OR (NOT 2 AND 7), cioè 5.

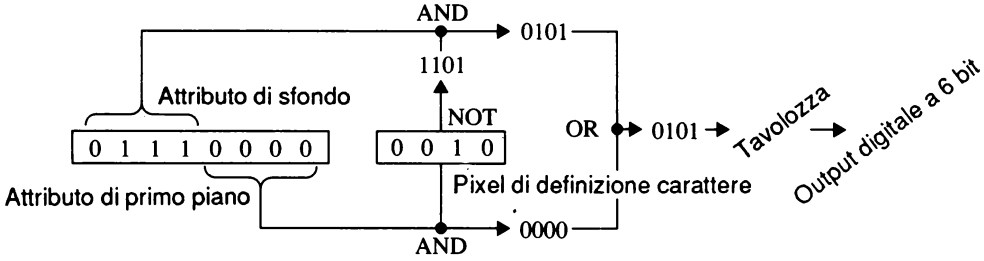


Figura 107. *Decodifica dell'attributo di colore di primo piano utilizzando definizioni di carattere basate su RAM (codici di carattere a 8 bit). Il valore di pixel nella definizione di carattere ed entrambi gli attributi del byte di attributo del carattere contribuiscono alla decodifica dell'attributo di primo piano.*

L'uso dei colori su scheda InColor è più semplice se caricate tutti i quattro piani di bit con identiche configurazioni di bit in modo che tutti i pixel delle definizioni di carattere abbiano il valore 0FH (1111B). A quel punto gli attributi di primo piano e di sfondo di un carattere dipendono unicamente dai valori del byte di attributo. In alternativa, potete specificare un attributo di primo piano di 0FH (1111B) ed un attributo di sfondo di 0 per ogni carattere del buffer del video. In questo caso, i colori visualizzati dipendono unicamente dai valori di pixel delle definizioni di carattere.

Un uso più pratico della RAM di definizione carattere della scheda InColor è quello di caricare ogni piano di bit con una diversa tabella di definizione caratteri. A quel punto ogni bit dell'attributo di primo piano di un carattere agisce da maschera per selezionare un diverso set di caratteri. Naturalmente, viene sempre generato un attributo di primo piano a 4 bit, come mostrato dalla Figura 107, quindi in effetti ogni set di caratteri viene associato al colore corrispondente al proprio piano di bit. Potete, naturalmente, visualizzare i set di caratteri in qualsiasi colore desideriate programmando i registri di tavolozza.

Per caricare separatamente i piani di bit, utilizzate il semibyte più significativo del registro di maschera di piano (18H) per proteggere in scrittura i piani di bit ogni volta che caricate un diverso set di caratteri. Ciò permette di utilizzare diversi attributi di primo piano per visualizzare i diversi set di caratteri. Ad esempio, se tutti i quattro piani di bit contengono diversi set di caratteri, potete selezionare ognuno dei quattro set di caratteri utilizzando gli attributi di primo piano 1, 2, 4 e 8.

Set di caratteri estesi

Tutti i sistemi di visualizzazione trattati in questo capitolo hanno una quantità sufficiente di RAM a disposizione del generatore di caratteri per memorizzare definizioni per più di 256 caratteri, quindi forniscono tutti un metodo che permette al generatore di caratteri di riconoscere i codici di carattere estesi composti da più dei soliti otto bit.

EGA e VGA

Su EGA e VGA, la solita gamma di 256 codici ASCII viene raddoppiata utilizzando il bit 3 del byte di attributo di un carattere per indicare una delle tabelle di definizione caratteri nell'area 2 (vedere Figura 108). In questo modo, possono essere visualizzati 512 diversi caratteri in una modalità alfanumerica.Normalmente, il valore del bit 3 del byte di attributo di un carattere non influenza il set di caratteri visualizzato. Il motivo è questo: il valore di questo bit seleziona uno dei due campi di bit del registro di selezione mappa caratteri del sequencer. A sua volta, il valore di ognuno di questi due campi di bit indica una delle tabelle di definizione caratteri disponibili in RAM. Quando il BIOS del video stabilisce una modalità di visualizzazione, carica un set di default di definizioni carattere nella prima tabella di definizione caratteri nell'area 2 ed azzera entrambi i campi di bit del registro di selezione mappa carattere.

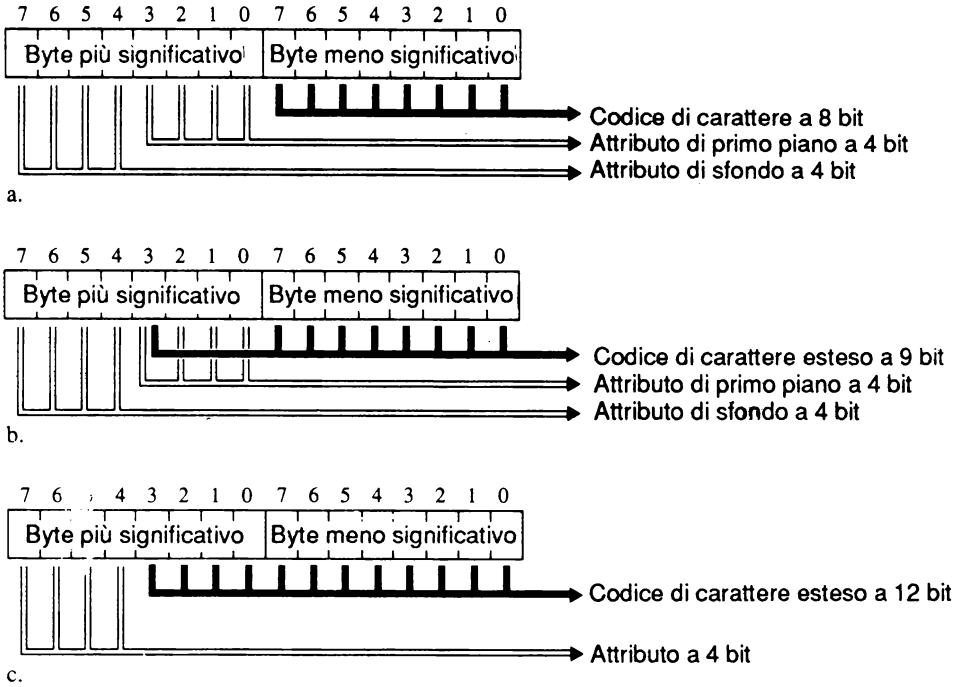


Figura 108. *Codici ed attributi di carattere. la figura 108a mostra il normale formato a 8 bit. La figura 108 bit mostra il formato esteso a 9 bit utilizzato su EGA,VGA e MCGA. la figura 108c mostra il formato esteso a 12 bit utilizzato su HGC+ e scheda iNColor .*

Di conseguenza, i caratteri alfanumerici di default vengono definiti dalle configurazioni di bit della prima tabella dell'area 2, a prescindere dal valore del bit 3 dei byte di attributo dei caratteri visualizzati.

La modifica del valore del registro di selezione mappa carattere, tuttavia, modifica le tabelle di definizione caratteri associate al bit 3 del byte di attributo di ogni carattere. Se due diversi valori appaiono nei campi di bit nel registro di selezione mappa carattere, il valore del bit 3 indica univocamente una delle due diverse tabelle di definizione caratteri.Ad esempio, nella Figura 109, il bit 3 è impostato a 1, quindi i bit 2,3 e 5 del registro di selezione mappa carattere indicano quale tabella di definizione caratteri utilizzare (questo esempio riguarda la VGA; su EGA, solo i bit 2 e 3 del valore del registro selezione mappa carattere sarebbero significativi).

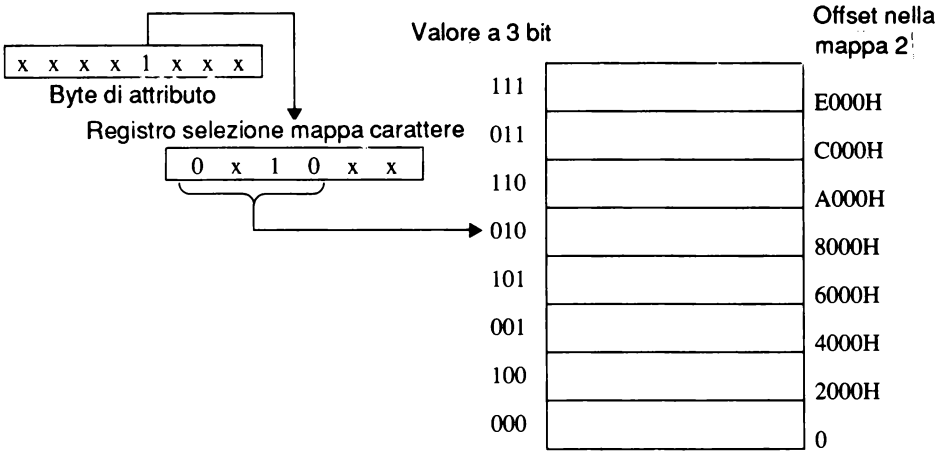


Figura 109. *Funzione del registro di selezione mappa carattere su VGA.*

Il Listato 98 illustra due metodi per l'aggiornamento di questo registro. Sebbene la tecnica dell'uso di una chiamata ad una funzione INT 10H richieda generalmente meno codice e sia più trasportabile, potreste preferire programmare direttamente il sequencer nelle applicazioni che richiedono un rapido passaggio tra set di caratteri.

```
; uso del BIOS del video

            mov     ax,1103h    ; AH := numero di funzione INT 10H
                                ; AL := 3
            mov     bl,CharMapValue ; BL := valore del registro
            int     10h             ;  selezione mappa carattere

; programmazione diretta del registro

            mov     dx,3C4h         ; DX := porta di I/O del sequencer
            mov     ax,100h         ; AH bit 1 := 0 (ripristino sincrono)
                                    ; AL := 0 (numero reg. ripristino
                                    ;  sequencer)
            cli                     ; disabilita interrupt
            out     dx,ax           ; ripristina sequencer
            mov     ah,CharMapValue ; AH := valore registro selezione
                                    ;  mappa carattere
            mov     al,3            ; AL := 3 (numero reg. selezione
                                    ;  mappa car.)
            out     dx,ax           ; aggiorna questo registro

            mov     ax,300h         ; AH bit 1:= 1 (azzera ripristino
                                    ;  sincrono)
                                    ; AL := 0 (numero registro ripristino)
            out     dx,ax           ; azzera ripristino
            sti                     ; abilita interrupt
```

Listato 98. *Programmazione del registro di selezione mappa carattere del sequencer su EGA e VGA.*

Se entrambi i campi di bit del registro di selezione mappa carattere contengono lo stesso valore, il valore del bit 3 del byte di attributo di un carattere non determina quale set di caratteri viene utilizzato. Se i campi di bit indicano diverse tabelle di definizione caratteri, allora il valore del bit 3 del byte di attributo di ogni carattere seleziona uno dei due diversi set di caratteri. Ricordate però che il bit 3 fa anche parte dell'attributo di primo piano a 4 bit di ogni carattere. Quando il bit 3 dell'attributo di primo piano di un carattere è impostato a 0, il colore visualizzato del carattere viene ricavato da uno dei primi otto registri di tavolozza (da 0000B a 0111B). Quando il bit 3 è impostato a 1, il colore viene ricavato da uno dei secondi otto registri di tavolozza (da 1000B a 1111B).

Di conseguenza, i due set di 256 caratteri selezionati dal bit 3 vengono visualizzati con due diversi set di otto valori di registri di tavolozza. Ciò risulta comodo se desiderate associare un particolare set di colori ad un set di caratteri. In caso contrario, potreste preferire caricare i secondi otto registri di tavolozza con lo stesso set di valori dei primi otto in modo che il valore del bit 3 del byte di attributo di un carattere non abbia alcuna influenza sul colore visualizzato. Un'altra tecnica è costituita dal mascheramento del bit 3 dell'attributo di primo piano azzerando il bit 3 del registro di abilitazione piano di colore del controller di attributo, come avviene nel Listato 99. Dal momento che il valore del registro di abilitazione piano colore maschera il valore dell'attributo a 4 bit, l'azzeramento del bit 3 in questo registro permette di far riferimento solo ai primi otto registri di tavolozza, a prescindere dal valore del bit 3 del byte di attributo del carattere.

```
mov     ax,1000h    ; AH := 10H (numero funzione INT 10H)
                    ; AL := 0 (imposta registro specificato)
mov     bx,0712h    ; BH := 0111b (valore abilitazione piano
                    ;  colore)
                    ; BL := 12H (numero reg abilitazione piano
                    ;  colore)
int     10h         ; aggiorna reg. abilitazione piano colore
```

Listato 99. *Azzeramento del bit 3 del registro abilitazione piano colore. Ciò impedisce al bit 3 del byte di attributo di un carattere di influenzare l'attributo visualizzato.*

MCGA

L'MCGA supporta i codici di carattere a 8 e a 9 bit con la stessa interfaccia BIOS dell'EGA e della VGA, sebbene l'implementazione hardware sia diversa. Su MCGA, le due tabelle di definizione caratteri selezionate dal bit 3 del byte di attributo di un carattere sono quelle contenute nelle due pagine di tipi di caratteri interne dell'MCGA. Sebbene possiate caricare le pagine di tipi di caratteri programmando il registro di interfaccia generatore di caratteri (12H), il registro del puntatore dei tipi di caratteri (13H) e il registro del numero di caratteri da caricare (14H) dell'MCGA, è più semplice utilizzare la funzione 11H di INT 10H con AL = 3.

Come su EGA e VGA, il bit 3 del byte di attributo di un carattere esegue un doppio lavoro come parte del codice di carattere a 9 bit oltre che come bit più significativo dell'attributo di primo piano del carattere. Se desiderate utilizzare gli stessi colori per entrambi i set di 256 caratteri, potete richiamare la funzione 10H di INT 10H per memorizzare gli stessi valori di set di colori nei secondi otto registri di colore del DAC del video esattamente come avviene per i primi otto. Potete anche richiamare la funzione 10H di INT 10H per mascherare il bit 3 alla decodifica dell'attributo alfanumerico (vedere Listato 99).

HGC+ e scheda InColor

Su HGC+ e sulle schede InColor, potete configurare il generatore di caratteri in modo che consideri i quattro bit meno significativi del byte di attributo di ogni carattere come parte del codice di carattere. Per fare ciò, impostate a 1 sia il bit 2 sia il bit 1 del registro Modalità-x.

L'uso dei codici di carattere a 12 bit permette di visualizzare tutti i caratteri definiti in qualsiasi area dei 48 KB della RAM del generatore di caratteri dell'adattatore Hercules. In pratica, potete considerare tutti i 48 KB della RAM del generatore di caratteri come se fossero un'unica tabella ininterrotta di definizione caratteri. Tuttavia, in alcune applicazioni, potreste trovare più comodo pensare alla RAM del generatore di caratteri come ad un set di dodici tabelle di 256 caratteri, dove i quattro bit più significativi del codice di carattere indicano una delle tabelle e gli otto bit meno significativi indicano una definizione all'interno della tabella.

Quando vengono utilizzati 12 bit come codice di carattere esteso, solo i bit da 4 a 7 del byte più significativo specificano l'attributo di un carattere (vedere Figura 108c). Gli

attributi che la Hercules ha assegnato a questi bit differiscono in qualche modo dai normali attributi di visualizzazione monocromatica (vedere Figura 110).

Bit attributo	Bit abil. lampeggiamento=1 (lampeggiamento abilitato)	Bit abil. lampeggiamento=0 (lampeggiamento disabilitato)
7	Evidenziato	Grassetto
6	Lampeggiamento	Inversione
5	Barrato	Non barrato
4	Sottolineato	Non sottolineato

Figura 110. *Set di attributi esteso su HGC+ e scheda InColor.*

CONSIGLIO

Quando utilizzate i codici di carattere a 12 bit su HGC+ e su scheda InColor, potete specificare la linea di scansione sulla quale devono apparire gli attributi di barratura e di sottolineatura. I bit da 0 a 3 del registro di sottolineatura (15H) controllano la posizione della sottolineatura. I bit da 0 a 3 del registro di barratura (16H) controllano la posizione della barratura. Sulla scheda InColor, potete anche controllare il colore visualizzato della sottolineatura e della barratura memorizzando un valore tra 1 e 0FH nei bit da 4 a 7 del corrispondente registro di controllo.

Come su HGC+, i codici di carattere a 12 bit sulla scheda InColor indicano le locazioni all'interno delle tabelle di definizione caratteri. Tuttavia la decodifica dell'attributo è più complicata sulla scheda InColor (vedere Figura 111). L'attributo di primo piano a 4 bit generato per ogni pixel di un carattere viene ricavato dalla combinazione dell'attributo a 4 bit del carattere con il valore del pixel della tabella di definizione caratteri.

Attributi compatibili MDA (registro eccezioni, bit 5 = 1)		
	Abilitazione lampegg. attiva	**Abilitazione lampegg. disattivata**
primo piano	(valore di pixel) OR (sfondo)	(valore di pixel) XOR (sfondo)
Sfondo	0 = se bit 7 dell'attributo = 0	0 se bit 6 dell'attributo = 0
	8 se bit 7 dell'attributo = 1	0FH se bit 6 dell'attributo = 1
Attributi colore (registro eccezioni, bit 5 = 0		
Primo piano	(valore di pixel) AND (NOT attributo)	
Sfondo	0	

Figura 111. *Decodifica dell'attributo di colore della scheda InColor utilizzando i codici di carattere a 12 bit.*

Come nel caso in cui si utilizzano i codici di carattere a 8 bit, la particolare interazione degli attributi di carattere con i valori di pixel nella tabella di definizione caratteri rende complesso il controllo sui colori. Per semplificare le cose, potete memorizzare le stesse definizioni di carattere in tutti i quattro piani di bit quando utilizzate la decodifica dell'attributo di colore; ciò permette all'attributo a 4 bit di ogni carattere di specificare tutti i 16 colori. Quando utilizzate gli attributi compatibili MDA, potete memorizzare le stesse configurazioni di bit nei piani di bit da 0 a 2 ed azzerare il piano di bit 3. Anche in questo caso, l'attributo a 4 bit di ogni carattere può controllare direttamente gli attributi visualizzati.

Se decidete di memorizzare diverse tabelle di definizione caratteri in ogni piano di bit, ognuno dei bit di attributo di un carattere può selezionare uno dei piani di bit. Anche in questo caso, dovreste impostare in modo accurato i registri di tavolozza per far sì che i caratteri dei diversi piani di bit vengano visualizzati con i colori appropriati.

Problemi di compatibilità con i codici di carattere estesi

La maggior parte dei programmi per PC e per PS/2, compreso il BIOS, l'MS-DOS e la maggior parte delle applicazioni disponibili in commercio, presumono che utilizziate codici di carattere ASCII a 8 bit. Questo significa che potete aggiornare la RAM del generatore di caratteri con un set di caratteri ASCII da 8 bit con un diverso tipo di caratteri, ma che non potete sfruttare i codici di caratteri a 9 o a 12 bit estesi supportati da IBM e da Hercules.

Se utilizzate l'interfaccia INT 10H per visualizzare i caratteri con codici di carattere estesi, dovete prestare attenzione quando utilizzate determinate funzioni BIOS ROM. Ad esempio, la funzione 0AH di INT 10H, che memorizza un codice di carattere a 8 bit nel buffer del video, non risulta molto utile per la scrittura di caratteri con codice di carattere esteso a 9 o a 12 bit. D'altro canto, potete utilizzare la funzione 9 di INT 10H, che gestisce una combinazione di codice e di attributo di carattere a 16 bit, per elaborare codici estesi ed attributi di caratteri.

Quando eseguite un'applicazione che utilizza codici di carattere estesi, potete incontrare problemi quando l'applicazione interagisce inavvertitamente con il software che non riconosce il diverso formato di attributo-carattere. Considerate cosa potrebbe accadere se un programma di utilità residente in RAM emergesse mentre l'applicazione è in corso di esecuzione senza "accorgersi" che stavate utilizzando i codici di caratteri estesi. Se il programma di utilità introducesse codici ed attributi di carattere a 8 bit nel buffer, il generatore di caratteri alfanumerici li interpreterebbe come codici ed attributi di caratteri estesi, e i risultati sarebbero catastrofici.

Modifica della matrice di carattere visualizzato

Esiste un'altra possibilità per personalizzare una tabella di definizione caratteri basata su RAM: potete controllare l'altezza della matrice di carattere all'interno della quale vengono visualizzati i caratteri. L'altezza della matrice di carattere visualizzato determina il numero delle righe di caratteri che appariranno sullo schermo. Ad esempio, uno schermo da 350 linee può contenere 43 righe di caratteri 8 per 8 ma solo 25 righe di caratteri 8 per 14.

Con tutti i sistemi di visualizzazione trattati nel corso del presente capitolo, potete variare l'altezza visualizzata dei caratteri alfanumerici programmando il controller del CRT in modo che visualizzi i caratteri con la stessa dimensione dei caratteri definiti nella RAM del generatore di caratteri. Di conseguenza, per visualizzare i caratteri 8 per 8 su uno schermo da 350 linee, introducete definizioni di carattere 8 per 8 nella RAM del generatore di caratteri, quindi programmate il CRTC in modo che visualizzi i caratteri alti 8 pixel.

Su EGA e VGA potete eseguire entrambe queste operazioni richiamando la funzione 11H di INT 10H, sebbene in alcune situazioni potreste preferire aggiornare le definizioni di carattere o programmare esplicitamente il CRTC. Gli adattatori Hercules, naturalmente, non hanno BIOS ROM, quindi dovete eseguire tutte le operazioni direttamente.

EGA

Considerate come potreste visualizzare 43 righe di caratteri 8 per 8 in una modalità alfanumerica EGA con una risoluzione verticale di 350 linee, come nel Listato 100. In questo esempio, la chiamata alla funzione 11H di INT 10H con AL = 12H copia il set di caratteri 8 per 8 della ROM (normalmente utilizzato nelle modalità video a 200 linee) nella prima delle quattro tabelle della mappa 2, quindi calcola i corretti valori di registro CRTC sulla base dei valori di POINTS e di ROWS nell'area dati di visualizzazione del BIOS.

```
; stabilisce modalità alfanumerica 80x25 (risoluzione verticale 350 linee)

            mov     ax,3        ; AH := 0 (numero funzione INT 10H)
            int     10h         ; AL := 3 (modalità a 16 colori 80x25)

; carica caratteri 8x8 del BIOS del video nel generatore di car.
;alfanumerici

            mov     ax,1112h    ; AH := numero funzione INT 10H
                                ; AL := set di caratteri 8x8 caricato
            mov     bl,0        ; BL := blocco da caricare
```

```
            int     10h         ; carica caratteri 8x8 in RAM

; imposta posizione cursore nella matrice del carattere

            mov     ax,40h
            mov     es,ax       ; ES ->i area dati BIOS del video
            mov     dx,es:[63h] ; DX := porta d'indirizzo CRTC da 0040:0063
                                ;  (3B4H o 3D4H)
            mov     ax,060Ah    ; AH := 6 (valore cursore inizio)
                                ; AL := 0AH (numero reg cursore inizio)
            out     dx,ax       ; aggiorna registro cursore inizio CRTC

            mov     ax,000Bh    ; AH := 0 (valore cursore fine)
                                ; AL := 0BH (numero reg cursore fine)
            out     dx,ax       ; aggiorna registro cursore fine CRTC

; usa routine alternativa stampa schermo del BIOS video

            mov     ah,12h      ; AH := numero funzione INT 10H
            mov     bl,20h      ; BL := numero sottofunzione
            int     10h         ; aggiorna vettore INT 5 (stampa schermo)
```

Listato 100. *Impostazione di una modalità alfanumerica 80 per 43 su EGA.*

La funzione 11H di INT 10H richiama la funzione 1 di INT 10H per impostare la posizione del cursore alfanumerico all'interno della matrice di carattere visualizzato. Come descritto nel Capitolo 3, la versione BIOS EGA della funzione 1 di INT 10H calcola questa posizione di cursore in modo errato, portando ad un cursore visualizzato in modo non corretto. Quindi, la routine del Listato 100 aggiorna direttamente i registri di cursore inizio e di cursore fine del CRTC.

CONSIGLIO

Se il vostro programma modifica il numero di righe di caratteri visualizzate, dovrebbe anche richiamare la funzione 12H di INT 10H per selezionare la routine alternativa di stampa schermo del BIOS EGA. Questa routine funziona in modo identico a quella del BIOS della piastra madre ad eccezione del fatto che utilizza il valore ROWS dell'area dati di visualizzazione per determinare il numero di righe da stampare (la routine del BIOS della piastra madre ignora ROWS e stampa sempre 25 righe).

VGA

Potete utilizzare la funzione 11H di INT 10H anche su VGA per stabilire una modalità alfanumerica con una matrice di carattere non standard (vedere Listato 101). Su VGA, impostate la risoluzione verticale della modalità video utilizzando la funzione 12H di INT 10H (con BL = 30H) prima di richiamare la funzione 11H. Inoltre, i calcoli di emu-

lazione cursore vengono eseguiti correttamente nel BIOS VGA, quindi non è necessario alcun codice supplementare.

```
; stabilisce modalità alfanumerica 80x25 con risoluzione verticale di 400
; linee

            mov     ax,1202     ; AH := 12h (numero funzione INT 10H)
                                ; AL := 2 (seleziona 400 linee di scansione)
            mov     bl,30h      ; numero di sottofunzione
            int     10h

            mov     ax,3        ; AH := 0 (numero funzione INT 10H)
            int     10h         ; AL := (modalità a 16 colori 80x25)

; carica caratteri 8x8 del BIOS del video nel generatore di car.
;alfanumerici

            mov     ax,1112h    ; AH := numero funzione INT 10H
                                ; AL := set di caratteri 8x8 caricato
            mov     bl,0        ; BL := blocco da caricare
            int     10h         ; carica caratteri 8x8 in RAM
```

Listato 101. *Impostazione di una modalità alfanumerica 80 per 50 su VGA.*

MCGA

L'MCGA può visualizzare solo caratteri con 2, 4, 6, 8, 10, 12, 14 o 16 linee di scansione (questo è un limite del controller della memoria dell'MCGA). Per modificare la matrice di carattere visualizzato, utilizzate la funzione 11H di INT 10H per caricare un nuovo set di caratteri nel generatore di caratteri. Successivamente, programmate il registro linee di scansione per carattere (09H) con un valore da 0 a 7; se il valore è *n*, il numero i linee di scansione visualizzato nella matrice di carattere è (*n*+1)x2. Il Listato 102 mostra come impostare una matrice di carattere 8 per 10 utilizzando la risoluzione verticale a 400 linee dell'MCGA per produrre 40 righe di 80 caratteri.

```
; stabilisce modalità alfanumerica 80x25

            mov     ax,3        ; AH := 0 (numero funzione INT 10H)
                                ; AL := 3 (modalità a 16 colori 80x25)
            int     10h

; azzera le configurazioni di bit nella RAM del generatore di caratteri

            mov     di,0A000h
            mov     es,di
            xor     di,di       ; ES:DI -> RAM del generatore di caratteri
            xor     ax,ax       ; AH := 0 (configurazione di bit)
                                ; AL := 0 (codice di carattere iniziale)
```

```
            mov     cx,256*16   ; CX := numero di word

L01:        stosw               ; memorizza codice carattere e azzera
            inc     al          ; AL := codice di carattere successivo
            loop    L01

; carica caratteri 8x8 del BIOS del video nel generatore di car.
;alfanumerici

            mov     ax,1102h    ; AH := numero funzione INT 10H
                                ; AL := set di caratteri 8x8 caricato
            mov     bl,0        ; BL := blocco da caricare
            int     10h         ; carica caratteri 8x8 in RAM

            mov     ax,1103h    ; AH := numero funzione INT 10H
                                ; AL := generatore caratteri caricato
            mov     bl,0        ; BL := blocchi da caricare
            int     10h         ; carica caratteri nel generatore di car.

; programma controller CRT per visualizzare caratteri 8x10

            mov     dx,3D4h     ; DX := indirizzo di porta di I/O MCGA
            mov     ax,409h     ; AL := 9 (numero registro)
                                ; AH := 4 (valore registro)
            out     dx,ax       ; aggiorna registro linee di scansione

            mov     al,0Ah      ; AL := 0AH (numero registro)
            out     dx,ax       ; aggiorna registro cursore inizio

            mov     al,0Bh      ; AL := 0BH (numero registro)
            out     dx,ax       ; aggiorna registro cursore fine

; aggiorna variabili di stato nel segmento dati BIOS del video

            mov     ax,40h
            mov     ds,ax       ; DX -> segmento dati BIOS del video

            mov     word ptr ds:[4Ch],80*40*2  ; aggiorna CRT_LEN
                                               ;  nell'area dati del BIOS
            mov     byte ptr ds:[84h],40-1     ; aggiorna ROWS
            mov     word ptr ds:[85h],10 ; aggiorna POINTS
```

Per alcuni valori del registro di linee di scansione per carattere, l'MCGA visualizza erroneamente la linea di scansione inferiore dello schermo. In particolare, quando il valore del registro di linee di scansione per carattere è 1, 3, 5 o 6, l'MCGA duplica parte della linea di scansione superiore sullo schermo nella parte inferiore dello schermo. Di conseguenza, dovreste in generale evitare di utilizzare questi valori per il registro di linee di scansione per carattere.

HGC+ e scheda InColor

Dovete programmare in modo esplicito il CRTC dell'HGC+ per modificare il numero di righe visualizzate con caratteri alfanumerici. La subroutine *SetHercCRTC* del Listato 105 illustra una tecnica guidata da tabella per l'impostazione dei parametri di temporizzazione verticale del CRTC per svariate dimensioni di caratteri. La Figura 112 riassume i parametri di temporizzazione del CRTC consigliati dalla Hercules per qualsiasi matrice di caratteri con altezza tra le 4 e le 16 linee di scansione oltre che per i caratteri che hanno una larghezza di 8 o 9 pixel.

	Larghezza della matrice di carattere	
Registro CRTC	**8pixel**	**9pixel**
00H	6DH	61H
01H	5AH	50H
02H	5CH	52H
03H	0FH	0FH

Registro	**Altezza della matrice di carattere (in pixel)**												
CRTC	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
04H	5CH	4AH	3DH	34H	2DH	28H	24H	20H	1DH	1BH	19H	17H	16H
05H	02H	00H	04H	06H	02H	01H	00H	07H	0AH	06H	06H	0AH	02H
06H	58H	46H	3AH	32H	2BH	26H	23H	1FH	1DH	1AH	19H	17H	15H
07H	59H	46H	3BH	33H	2CH	27H	23H	20H	1DH	1BH	19H	17H	16H

Figura 112. *Parametri di temporizzazione del CRTC per altezza e larghezza della matrice di caratteri alfanumerici (HGC+ e scheda InColor).*

Su scheda InColor, le tecniche per la modifica della matrice di carattere visualizzato sono analoghe a quelle utilizzate su HGC+. Anche i valori introdotti nei registri CRTC per ogni possibile matrice di carattere sono identici.

Esempi di programmazione

Le routine delle pagine seguenti riuniscono le tecniche di programmazione per la modifica della matrice di carattere visualizzato su EGA (vedere Listato 103), su VGA (vedere Listato 104) e su HGC+ e scheda InColor (vedere Listato 105). In ogni caso trattato, la funzione *AlphaModeSet()* imposta il generatore di caratteri alfanumerici e il CRTC per conformarsi alle dimensioni della matrice di carattere specificata ed alla dimensione del codice di carattere.

```
                TITLE   'Listato 103'
                NAME    AlphaModeSet
                PAGE    55,132

;
; Nome:         AlphaModeSet
;
;  Programma il CRTC nelle modalità alfanumeriche a 80 colonne EGA
;
; Chiamante:    Microsoft C:
;
;               azzera AlphaModeSet(w,h,c);
;
;               int     w;          /* larghezza della matrice di carattere */
;               int     h;          /* altezza della matrice di carattere */
;               int     c;          /* dimensione del codice di carattere*/



ARGw            EQU     byte ptr [bp+4] ; deve essere larga 8 o 9 pixel
ARGh            EQU     byte ptr [bp+6] ; deve essere alta da 2 a 32 pixel
ARGc            EQU     byte ptr [bp+8] ; deve essere 8 o 9 bit

CRT_MODE        EQU     49h             ; indirizzi nell'area dati di
                                        ;  visualizzazione del BIOS
CRT_COLS        EQU     4Ah
ADDR_6845       EQU     63h

DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                PUBLIC _AlphaModeSet
_AlphaModeSet PROC      near

                push    bp              ; mantiene registri del chiamante
                mov     bp,sp
                push    si

; programma il CRTC

                mov     bx,40h
                mov     es,bx           ; ES := segmento dati BIOS del video

                mov     bl,ARGw         ; BL := larghezza carattere
                mov     bh,ARGh         ; BH := altezza carattere
                call    SetCRTC

; programma il sequencer ed il controller di attributo per 8 o 9 punti
; per carattere

                mov     dx,3C4h
                mov     ax,0100h        ; AH bit 1 := 0 (reset sincrono)
                                        ; AL := 0 (numero registro reset)
                cli                     ; disabilita interrupt
                out     dx,ax           ; reset sincrono del sequencer
```

```
                mov     bx,1        ; BH,BL := valori per caratteri larghi 8:
                                    ;  BH := 0 (valore per Scorr.Pixel oriz)
                                    ;  BL := 1 (valore per modalità
                                    ;  sincronizzazione)
                cmp     ARGw,8
                je      L01         ; salta se caratteri larghi 8

                mov     bx,0800h    ; BH,BL := valori per caratteri larghi 9

L01:            mov     ah,bl       ; AH := valore per reg modalità
                                    ;  sincronizzazione
                mov     al,1        ; AL := numero reg modalità
                                    ;  sincronizzazione
                out     dx,ax       ; programma il sequencer

                mov     ax,0300h    ; AH := 3 (disabilita ripristino)
                                    ; AL := 0 (numero registro sequencer)
                out     dx,ax       ; disabilita reset sequencer
                sti                 ; abilita interrupt

                mov     bl,13h      ; BL := numero registro Scorr.Pixel
                                    ;  orizzontale
                mov     ax,1000h    ; AH := 10H (numero funzione INT 10H)
                                    ; AL := 0 (imposta registro specificato)
                int     10h         ; programma controller attributo

; programma il controller di attributo per codici di carattere da 8 o 9
; bit

                mov     ax,1000h    ; AH := 10H (numero funzione INT 10H)
                                    ; AL := 0 (imposta registro specificato)
                mov     bx,0F12h    ; BH := 0FH (valore abilitazione
                                    ;  piano colore)
                                    ; BL := 12H (# reg abilitazione
                                    ;  piano colore)
                cmp     ARGc,8
                je      L02         ; salta se codici carattere a 8 bit

                mov     bh,7        ; BH bit 3 := 0 (ignora bit 3 di tutti
                                    ;  gli attributi)
L02:            int     10h         ; aggiorna registro abilitazione
                                    ;  piano colore

; aggiorna area dati di visualizzazione BIOS

                cmp     byte ptr es:[CRT_MODE],7
                jne     L03         ; salta se non è modalità monocromatica

                mov     ax,720      ; AX := pixel visualizzati per riga
                div     ARGw        ; AL := colonne di caratteri visualizzate
                mov     es:[CRT_COLS],al

L03:            pop     si
                pop     bp
                ret

_AlphaModeSet ENDP
```

```
SetCRTC         PROC    near        ; Chiamante:     BH = altezza carattere
                                    ;                BL = larghezza carattere

                push    dx
                mov     dx,es:[ADDR_6845]     ; porta di I/O del CRTC

; stabilisce temporizzazione verticale del CRTC e posizione del cursore
; nella matrice di carattere

                push    bx          ; mantiene altezza e larghezza
                mov     ax,1110h    ; AH := 11H (numero funzione INT 10H)
                                    ; AL := 0 (alfabeto utente caricato)
                xor     cx,cx       ; CX := 0 (non memorizza alcun carattere)
                int     10h         ; richiama BIOS per programmare il CRTC
                                    ;  per dimensione verticale dei caratteri

                pop     ax          ; AH := altezza caratteri
                push    ax          ; mantiene altezza e larghezza
                sub     ah,2        ; AH := linea di scansione iniziale
                                    ;
                                    ;  del cursore
                mov     al,0Ah      ; AL := 0AH (numero reg cursore inizio)
                out     dx,ax       ; aggiorna registro cursore inizio del CRTC

                mov     ax,000Bh    ; AH := 0 (valore cursore fine)
                                    ; AL := 0BH (numero reg cursore fine)
                out     dx,ax       ; aggiorna registro cursore fine del CRTC

; stabilisce temporizzazione orizzontale del CRTC

                pop     bx          ; BX := altezza e larghezza carattere
                cmp     byte ptr es:[CRT_MODE],7
                jne     L10         ; esce se non è modalità monocromatica

                xor     bh,bh       ; BX := larghezza carattere
                sub     bl,8        ; BX := 0 o 1
                neg     bx          ; BX := 0 o 0FFFFH
                and     bx,14       ; BX := 0 o 14 (offset nella tabella)
                mov     si,bx       ; SI := offset nella tabella

                add     si,offset DGROUP:HorizParms  ; DS:SI -> parametri
                call    UpdateCRTC

L10:            pop     dx
                ret

SetCRTC         ENDP


UpdateCRTC      PROC    near        ; Chiamante:     DX = porta indirizzo CRTC
                                    ;                DS:SI -> parametri
                                    ; distrugge:     AX,CX

                mov     cx,7        ; CX := numero registri da aggiornare

L20:            lodsw               ; AH := dati per registro del CRTC in AL
                out     dx,ax       ; aggiorna il registro
                loop    L20
```

```
                ret

UpdateCRTC      ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

HorizParms      DW      6C00h,5901h,6002h,2403h,5B04h,6A05h,2D13h; larg. 8
                DW      6000h,4F01h,5602h,3A03h,5104h,6005h,2813h; larg. 9

_DATA           ENDS

                END
```

Listato 103. *Programmazione della dimensione del carattere alfanumerico su EGA.*

```
                TITLE   'Listato 104'
                NAME    AlphaModeSet
                PAGE    55,132

;
; Nome:         AlphaModeSet
;
;  Programma il CRTC nelle modalità alfanumeriche a 80 colonne VGA
;
; Chiamante:    Microsoft C:
;
;                       azzera AlphaModeSet(w,h,c);
;
;                       int   w;    /* larghezza della matrice di carattere */
;                       int   h;    /* altezza della matrice di carattere */
;                       int   c;    /* dimensione del codice di carattere */


ARGw            EQU     byte ptr [bp+4] ; deve essere larga 8 o 9 pixel
ARGh            EQU     byte ptr [bp+6] ; deve essere alta da 2 a 32 pixel
ARGc            EQU     byte ptr [bp+8] ; deve essere 8 o 9 bit

CRT_COLS        EQU     4Ah         ; indirizzi nell'area dati
                                    ;  di visualizz. del BIOS
ADDR_6845       EQU     63h

DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP
```

```
                PUBLIC _AlphaModeSet
_AlphaModeSet PROC   near

                push   bp          ; mantiene registri chiamante
                mov    bp,sp
                push   si

; programma il CRTC

                mov    bx,40h
                mov    es,bx       ; ES := segmento dati del BIOS del video

                mov    bl,ARGw     ; BL := larghezza carattere
                mov    bh,ARGh     ; BH := altezza carattere
                call   SetCRTC

; programma il sequencer ed il controller di attributo per 8 o 9 punti
;per carattere

                mov    dx,3C4h
                mov    ax,0100h    ; AH bit 1 := 0 (reset sincrono)
                                   ; AL := 0 (numero registro reset)
                cli                ; disabilita interrupt
                out    dx,ax       ; reset sincrono sequencer

                mov    bx,1        ; BH,BL := valori per caratteri larghi 8:
                                   ;  BH := 0 (valore per Scorr.Pixel
                                   ;  orizzontale)
                                   ;  BL := 1 (valore per modalità
                                   ;  sincronizzazione)
                cmp    ARGw,8
                je     L01         ; salta se caratteri sono larghi 8

                mov    bx,0800h    ; BH,BL := valori per caratteri larghi 9

L01:            mov    ah,bl       ; AH := valore per reg modalità
                                   ;  sincronizzazione
                mov    al,1        ; AL := numero reg modalità
                                   ;  sincronizzazione
                out    dx,ax       ; programma il sequencer

                mov    ax,0300h    ; AH := 3 (disabilita reset)
                                   ; AL := 0 (numero registro sequencer)
                out    dx,ax       ; disabilita reset sequencer
                sti                ; abilita interrupt

                mov    bl,13h      ; BL := numero reg Scorr.Pixel orizzontale
                mov    ax,1000h    ; AH := 10H (numero funzione INT 10H)
                                   ; AL := 0 (imposta registro specificato)
                int    10h         ; programma controller attributo

; programma controller attributo per codici carattere a 8 o 9 bit

                mov    ax,1000h    ; AH := 10H (numero funzione INT 10H)
                                   ; AL := 0 (imposta registro specificato)
                mov    bx,0F12h    ; BH := 0FH (valore abilitazione
                                   ;  piano colore)
```

```
                                ; BL := 12H (# reg abilitazione
                                ;  piano colore)
            cmp     ARGc,8
            je      L02         ; salta se codici di carattere a 8 bit

            mov     bh,7        ; BH bit 3 := 0 (ignora bit 3 di tutti
                                ;  gli attributi)
L02:        int     10h         ; aggiorna registro abilitazione
                                ;  piano colore

; aggiorna area dati BIOS del video

            mov     ax,720      ; AX := pixel visualizzati per riga
            div     ARGw        ; AL := colonne di carattere
                                ;  visualizzate
            mov     es:[CRT_COLS],al

            pop     si
            pop     bp
            ret

_AlphaModeSet ENDP



SetCRTC     PROC    near        ; Chiamante:      BH = altezza carattere
                                ;                 BL = larghezza carattere

            push    dx
            mov     dx,es:[ADDR_6845]     ; porta di I/O del CRTC

; stabilisce temporizzazione verticale del CRTC e posizione del cursore
;nella matrice di carattere

            push    bx          ; mantiene altezza e larghezza caratteri
            mov     ax,1110h    ; AH := 11H (numero funzione INT 10H)
                                ; AL := 0 (alfabeto utente caricato)
            xor     cx,cx       ; CX := 0 (non memorizza alcun carattere)
            int     10h         ; richiama BIOS per programmare il CRTC
            pop     bx

; abilita operazioni di scrittura di I/O sui registri del CRTC

            mov     al,11h      ; AL := numero reg ritracciamento
                                ;  verticale fine
            out     dx,al
            inc     dx
            in      al,dx       ; AL := valore corrente di questo
                                ;  registro
            dec     dx

            mov     ah,al       ; AH := valore corrente
            mov     al,11h      ; AL := numero registro
            push    ax          ; salva su stack

            and     ah,01111111b    ; azzera bit 7
            out     dx,ax           ; aggiorna questo registro
```

```
; stabilisce temporizzazione orizzontale del CRTC

            xor     bh,bh       ; BX := larghezza carattere
            sub     bl,8        ; BX := 0 o 1
            neg     bx          ; BX := 0 o 0FFFFH
            and     bx,14       ; BX := 0 o 14 (offset nella tabella)
            mov     si,bx       ; SI := offset nella tabella

            add     si,offset DGROUP:HorizParms  ; DS:SI -> parametri
            call    UpdateCRTC

; protegge in scrittura i registri del CRTC

            pop     ax          ; AX := dati prec. del reg ritracc.
                                ;  vertic. fine
            out     dx,ax       ; ripristina questo registro

            pop     dx
            ret

SetCRTC     ENDP


UpdateCRTC  PROC    near        ; Chiamante:     DX = porta indirizzo CRTC
                                ;                DS:SI -> parametri
                                ; distrugge:     AX,CX

            mov     cx,7        ; CX := numero dei registri da aggiornare

L10:        lodsw               ; AH := dati per registro CRTC in AL
            out     dx,ax       ; aggiorna il registro
            loop    L10

            ret

UpdateCRTC  ENDP

_TEXT       ENDS


_DATA       SEGMENTword public 'DATA'

HorizParms  DW      6A00h,5901h,5A02h,8D03h,6304h,8805h,2D13h  ; larg. 8
            DW      5F00h,4F01h,5002h,8203h,5504h,8105h,2813h  ; larg. 9

_DATA       ENDS


            END
```

Listato 104. *Programmazione della dimensione del carattere alfanumerico su VGA.*

```
              TITLE   'Listato 105'
              NAME    AlphaModeSet
              PAGE    55,132

;
; Nome:        AlphaModeSet
;
; Funzione:    Programma il CRTC nelle modalità alfanumeriche su HGC+ o
;scheda InColor
;
; Chiamante:   Microsoft C:
;
;                     azzera AlphaModeSet(w,h,c);
;
;                     int   w;    /* larghezza della matrice del carattere */
;                     int   h;    /* altezza della matrice di carattere */
;                     int   c;    /* dimensione del codice di carattere */
;

ARGw          EQU     byte ptr [bp+4]       ; deve essere larga 8 o 9 pixel
ARGh          EQU     byte ptr [bp+6]       ; deve essere alta da 4 a 16 pixel
ARGc          EQU     byte ptr [bp+8]       ; deve essere 8 o 12 bit

CRT_COLS      EQU     4Ah
CRT_LEN       EQU     4Ch
CRT_MODE_SET  EQU     65h
ROWS          EQU     84h

DGROUP        GROUP   _DATA

_TEXT         SEGMENTbyte public 'CODE'
              ASSUME cs:_TEXT,ds:DGROUP

              PUBLIC _AlphaModeSet
_AlphaModeSet PROC    near

              push    bp              ; mantiene registri del chiamante
              mov     bp,sp
              push    ds
              push    si



; Imposta commutatore configurazione per portare RAM a partire da
;B000:4000 nella mappa di memoria

              mov     dx,3BFh         ; DX := porta commutatore
                                      ;  configurazione
              mov     al,1            ; AL bit 1 := 0 (esclude secondi 32K
                                      ;  del buffer del video)
                                      ; AL bit 0 := 1 (rende indirizzabile
              out     dx,ax           ; RAM a B000:4000)

; Cancella lo schermo per evitare interferenze durante la programmazione
; del CRTC
```

```
            mov     dx,3B8h         ; DX := porta reg. di controllo
                                    ;  modalità CRTC
            xor     al,al           ; AL bit 3 := 0 (disabilita
                                    ;  segnale video)
            out     dx,al           ; cancella lo schermo

; Programma il CRTC

            mov     bh,ARGw         ; BH := larghezza carattere
            mov     bl,ARGh         ; BL := altezza carattere
            call    SetHercCRTC

; Imposta xModeReg (registro Modalità-x)

            mov     dx,3B4h         ; DX := porta indirizzo CRTC
            mov     ax,114h         ; AH bit 0 := 1 (abilita generatore
                                    ;  caratteri basato su RAM)
                                    ; AL := 14h (numero xModeReg)
            cmp     ARGw,9
            je      L01             ; salta se caratteri sono larghi 9

            or      ah,2            ; AH bit 1 := 1 (caratteri larghi 8)

L01:        cmp     ARGc,8
            je      L02             ; salta se codici di carattere
                                    ;  sono a 8 bit

            or      ah,4            ; AH bit 2 := 1 (codici di carattere
                                    ;  a 12 bit)

L02:        out     dx,ax           ; aggiorna il registro

; aggiorna area dati del BIOS del video

            mov     ax,40h
            mov     ds,ax           ; DS := segmento dati del BIOS
                                    ;  del video

            mov     ax,720          ; AX := pixel visualizzati per riga
            div     ARGw            ; AL := colonne di carattere
                                    ;  visualizzate
            mov     ds:[CRT_COLS],al

            mov     ax,350          ; AX := numero linee di scansione
                                    ;  visualizzate
            div     ARGh            ; AL := righe di caratteri
                                    ;  visualizzate
            dec     al              ; AL := (righe di caratteri) - 1
            mov     ds:[ROWS],al

            inc     al
            mul     byte ptr ds:[CRT_COLS]
            shl     ax,1            ; AX := righe * colonne * 2
            mov     ds:[CRT_LEN],ax

; riabilita schermo ed esce
```

```
                mov     dx,3B8h         ; DX := porta controllo modalità
                                        ;  del CRT
                mov     al,ds:[CRT_MODE_SET] ; ripristina valore precedente
                out     dx,al

                pop     si
                pop     ds
                pop     bp
                ret

_AlphaModeSet ENDP


SetHercCRTC     PROC    near            ; Chiamante:BH = larghezza carattere
                                        ;           BL = altezza carattere

                push    dx
                mov     dx,3B4h         ; DX := porta reg indirizzo
                                        ;  del CRTC 3B4h

; stabilisce posizione cursore nella matrice del carattere

                mov     ah,bl
                dec     ah              ; AH := valore per reg linea
                                        ;  di scansione max
                mov     al,9            ; AL := numero registro linea
                                        ;  di scansione max
                out     dx,ax

                mov     al,0Bh          ; AL := numero reg cursore fine
                out     dx,ax           ; imposta cursore a fine ultima
                                        ;  linea della matrice di carattere

                sub     ax,101h         ; AH := da seconda a ultima linea
                                        ; AL := 0AH (numero reg cursore
                                        ;  inizio)
                out     dx,ax           ; imposta cursore a inizio su
                                        ;  seconda a ultima linea

; calcola offset nelle tabelle dei parametri

                sub     bx,0804h        ; BH := 0 o 1
                                        ; BL := da 0 a 12
                add     bx,bx
                add     bx,bx           ; BH := 0 o 4
                                        ; BL := da 0 a 48

; stabilisce temporizzazione orizzontale del CRTC

                push    bx              ; mantiene BX
                mov     bl,bh
                xor     bh,bh           ; BX := 0 o 4
                add     bx,offset DGROUP:HorizParms  ; DS:BX -> parametri

                mov     al,0            ; AL := primo reg del CRTC
                                        ;  da aggiornare
```

```
                call    UpdateCRTC

; stabilisce temporizzazione verticale

                pop     bx
                xor     bh,bh           ; BX := da 0 a 48
                add     bx,offset DGROUP:VertParms  ; DS:BX - parametri

                mov     al,4            ; AL := primo reg del CRTC
                                        ;  da aggiornare
                call    UpdateCRTC

                pop     dx              ; ripristina DX
                ret

SetHercCRTC     ENDP


UpdateCRTC      PROC    near            ; Chiamante:AL = numero primo reg
                                        ;           DX = porta indirizzo CRTC
                                        ;           DS:BX - parametri
                                        ; Distrugge:AX,CX

                mov     cx,4            ; CX := numero di registri
                                        ;  da aggiornare

L10:            mov     ah,[bx]         ; AH := dati per registro
                                        ;  del CRTC in AL
                out     dx,ax           ; aggiorna il registro
                inc     ax              ; AL := numero registro successivo
                inc     bx              ; DS:BX - valore successivo
                                        ;  nella tabella
                loop    L10

                ret

UpdateCRTC      ENDP

_TEXT           ENDS

_DATA           SEGMENTword public 'DATA'

HorizParms      DB      6Dh,5Ah,5Ch,0Fh       ; larga 8 pixel
                DB      61h,50h,52h,0Fh       ; larga 9 pixel

VertParms       DB      5Ch,02h,58h,59h       ; alta 4 linee di scansione
                DB      4Ah,00h,46h,46h       ;      5
                DB      3Dh,04h,3Ah,3Bh       ;      6
                DB      34h,06h,32h,33h       ;      7
                DB      2Dh,02h,2Bh,2Ch       ;      8
                DB      28h,01h,26h,27h       ;      9
                DB      24h,00h,23h,23h       ;      10
                DB      20h,07h,1Fh,20h       ;      11
                DB      1Dh,0Ah,1Dh,1Dh       ;      12
                DB      1Bh,06h,1Ah,1Bh       ;      13
                DB      19h,06h,19h,19h       ;      14
                DB      17h,0Ah,17h,17h       ;      15
                DB      16h,02h,15h,16h
```

```
_DATA          ENDS

               END
```

Listato 105. *Programmazione della dimensione del carattere alfanumerico su HGC+ e su scheda InColor.*

Finestre grafiche in modalità alfanumeriche

Quando aggiornate una tabella di definizione caratteri residente in RAM, modificate l'aspetto di qualsiasi carattere visualizzato utilizzando le definizioni inserite. Il contenuto della parte visualizzata del buffer del video non deve essere aggiornato, quindi potete sfruttare questa caratteristica delle definizioni di carattere basate su RAM per visualizzare immagini grafiche indirizzabili a pixel in una modalità alfanumerica, visualizzando il testo alla massima velocità e nel contempo includendo immagini grafiche pixel per pixel sullo stesso schermo.

La tecnica è simile sia per i sottosistemi IBM sia per quelli Hercules. Riempite a mosaico un'area dello schermo con una sequenza di caratteri il cui attributo selezioni una tabella di definizione caratteri che contenga l'immagine grafica (vedere Figura 113). L'immagine grafica viene creata e modificata aggiornando le definizioni di carattere appropriate nella tabella. Potete considerare la tabella di definizione caratteri come una sorta di buffer grafico virtuale e potete accedere ai pixel individuali all'interno di essa esattamente come fareste nelle modalità grafiche normali.

Su scheda InColor, potete anche specificare il valore di ogni pixel individuale memorizzato nella tabella di definizione caratteri come se utilizzaste la modalità grafica a 16 colori 720 per 348. Sugli altri sottosistemi, tuttavia, viene utilizzata solo una mappa di memoria per definizioni carattere, quindi non avete il controllo sull'attributo pixel per pixel. Al contrario, i pixel della tabella di definizione caratteri hanno il valore 0 o 1; gli attributi con i quali i codici di carattere vengono memorizzati nel buffer del video determinano l'aspetto dei pixel.

Il Listato 106 illustra la tecnica per la produzione di una finestra grafica a mosaico in una modalità alfanumerica a 80 colonne su EGA e VGA. La prima parte del programma crea la finestra a mosaico memorizzando i secondi 128 caratteri ASCII in quattro righe di 32 all'inizio del buffer del video (cioè, nell'angolo superiore sinistro dello schermo). Successivamente, il programma cancella la finestra impostando a 0 le seconde 128 definizioni di carattere.

Per aggiornare un pixel nella finestra, la subroutine *SetPixel()* calcola un offset di byte nella tabella di definizione caratteri che corrisponda alla locazione di pixel nella finestra a mosaico. Come nelle modalità grafiche, la routine accede ad ogni pixel individuale con una maschera di bit.

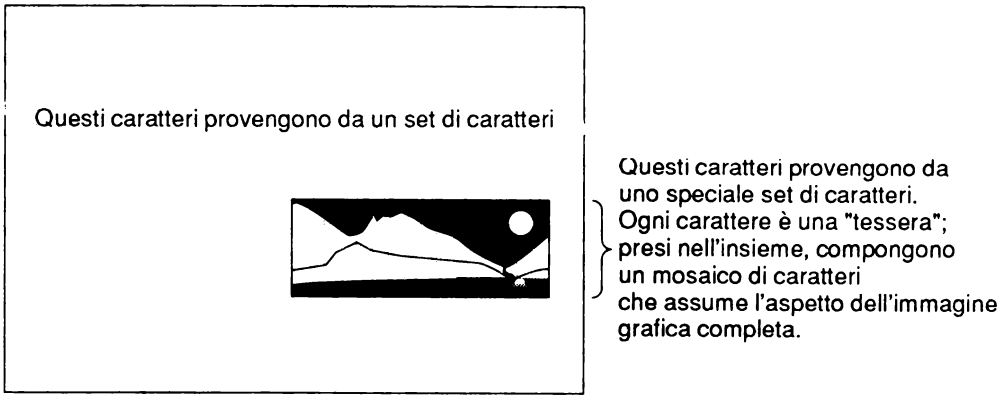


Figura 113. *Una finestra grafica composta a mosaico in una modalità alfanumerica.*

```
/* Listato 106 */

#define       Points            14  /* linee di scansione visualizzate
                                       per car. */
#define       StartCharCode     0x80/* primo codice di caratt.
                                       nella "finestra" */
#define       CGenDefSize       32  /* (usa 16 per Hercules) */

char far *CRT_MODE = 0x00400049;     /* numero modalità video del BIOS */
int  far *CRT_COLS = 0x0040004A;     /* caratteri per riga */

char far *VideoBuffer;               /* puntatore al buffer del video */
char far *CharDefTable = 0xA0000000; /* puntatore alla RAM di */
                                     /*  definizione caratt. */
                                     /* (usa 0xB0004000 per Hercules) */
main()
{
              int    i;
              int    CharCode;
              int    CharOffset;
              int    CharScanLine;
              int    CharDefOffset;
              int    Row,Column;



              /* stabilisce modalità alfanumerica */

              if (*CRT_MODE == 7)
                                 /* imposta puntatore buffer del video */
                VideoBuffer = 0xB0000000;
              else
```

```
    VideoBuffer = 0xB8000000;

  AlphaModeSet( 8, Points, 8 );


  /* stabilisce una finestra grafica a mosaico nell'angolo */
  /*  superiore sinistro */

  CharCode = StartCharCode;

  for ( Row=0; Row<4; Row++ )
    for ( Column=0; Column <32; Column++ )
    {
      CharOffset = (Row*(*CRT_COLS) + Column) * 2;
      VideoBuffer[CharOffset] = CharCode++;
    }


  /* cancella la finestra */

  CGenModeSet();     /* rende indirizzabile la RAM del */
                     /*  generatore di caratteri */

  for (CharCode=StartCharCode; CharCode < 256; CharCode++ )
    for ( CharScanLine=0; CharScanLine < Points ; CharScanLine++ )
    {
      CharDefOffset = CharCode*CGenDefSize + CharScanLine;
      CharDefTable[CharDefOffset] = 0;
    }


  /* traccia alcune linee */

  for ( i=0; i; i++ )                /* linee orizzontali */
  {
    SetPixel( i, 0 );
    SetPixel( i, 4*Points-1 );
  }

  for ( i=0; i*Points-1; i++ )       /* linee verticali */
  {
    SetPixel( 0, i );
    SetPixel( 255, i );
  }

  for( i=0; i        *4; i++ )       /* linee diagonali */
  {
    SetPixel( i, i );
    SetPixel( 255-i, i );
  }

  CGenModeClear();       /* ripristina modalità alfanum. */

}

SetPixel( x, y )
int        x,y;                    /* coordinate di pixel */
{
```

```
        int     CharCode;
        int     CharScanLine;
        int     BitMask;
        int     CharDefOffset;


        CharCode = StartCharCode + (y/Points)*32 + x/8;
        CharScanLine = y % Points;   /* Punti y MOD */
        BitMask = 0x80 > (x % 8);    /* 10000000b SHR (x MOD 8) */

        CharDefOffset = CharCode*CGenDefSize + CharScanLine;
        CharDefTable[CharDefOffset] |= BitMask;
                                     /* effettua l'OR del pixel */
}
```

Listato 106. *Creazione di una finestra grafica a mosaico su EGA o VGA.*

HGC+ e scheda InColor

Chiaramente, la dimensione di una finestra grafica a mosaico è ristretta se utilizzate codici di carattere a 8 bit perché il set di caratteri ASCII a 8 bit contiene solo 256 caratteri. Se configurate un adattatore Hercules per codici di carattere a 12 bit, però, potete creare finestre a mosaico molto più grandi senza esaurire i codici di carattere. Inoltre, potete creare finestre più grandi visualizzando caratteri più alti (aumentando cioè l'altezza della matrice di carattere visualizzato). Naturalmente, se utilizzate caratteri più alti, diminuite il numero di righe di testo visualizzabili contemporaneamente; questo potrebbe costituire un inconveniente in alcune applicazioni.

Potete utilizzare tecniche di programmazione analoghe per la modalità grafica alfanumerica su adattatori Hercules e su sottosistemi IBM. Ad esempio, il Listato 106 può essere modificato per essere usato con HGC+ e scheda InColor cambiando i valori di *CGenDefSize* e di *CharDefTable* ed eliminando le chiamate alle funzioni *CGenModeSet()* e *CGenModeClear()*.

CONSIGLIO

Nello definire una finestra grafica su scheda Hercules, evitate di utilizzare una matrice di carattere larga 9 pixel. Dal momento che il nono pixel (quello posto all'estrema destra) di ogni carattere è effettivamente una copia generata dall'hardware dell'ottavo punto, non potete controllarlo individualmente aggiornando la tabella di definizione caratteri.

EGA e VGA

Su EGA e VGA, potete creare finestre grafiche a mosaico più grandi se utilizzate codici di carattere estesi a 9 bit. Ad esempio, potreste dedicare una tabella di definizione di 256 caratteri ai caratteri di testo ed una seconda tabella di definizione caratteri ai caratteri di "tessera" di mosaico grafico. Ciononostante, l'EGA e la VGA sono comunque limitate alla visualizzazione contemporanea di non più di 512 diversi caratteri, quindi la finestra grafica a mosaico più grande è molto più piccola di quella visualizzabile su un adattatore Hercules.

CONSIGLIO

Quando aggiornate i pixel della finestra a mosaico, dovreste ridurre il numero di volte in cui il programma ripristina il sequencer (ad esempio, nelle routine *CGenModeSet()* e *CGenModeClear()*). Se ripristinate il sequencer ogni volta che aggiornate un pixel, potreste creare delle interferenze con lo schermo (la sincronizzazione del reset del sequencer con l'intervallo di ritracciamento verticale può eliminare questa interferenza ma può anche diminuire enormemente la velocità di un programma). Se dovete tracciate un'immagine grafica complessa contenente molti pixel, tracciate l'intera figura in una volta sola come nel Listato 106.

MCGA

Le tabelle di definizione caratteri della RAM del generatore di caratteri dell'MCGA vengono formattate in modo diverso da quelle dell'EGA e della VGA, quindi una routine che manipola i pixel nella RAM del generatore di caratteri deve indirizzare in modo diverso le tabelle (vedere Listato 107). Inoltre, ricordate che lo schermo non rispecchia i cambiamenti alle tabelle di definizione caratteri dell'MCGA fino a quando non caricate le pagine di tipi di caratteri del generatore di caratteri (vedere Listato 106).

```
SetPixel( x, y )
int           x,y;                        /* coordinate di pixel */
{
              int     CharCode;
              int     CharScanLine;
              int     BitMask;
              int     CharDefOffset;


              /* la finestra è larga 32 caratteri */
              CharCode = StartCharCode + (y/Points)*32 + x/8;
```

```
            CharScanLine = y % Points;   /* Punti y MOD */
            BitMask = 0x80 > >(x % 8);   /* 10000000b SHR (x MOD 8) */

            CharDefOffset = CharCode*2 + CharScanLine*512 + 1;
            CharDefTable[CharDefOffset] |= BitMask;  /* effettua l'OR
                                              del pixel */
}
```

Listato 107. *Una routine per impostare i pixel in una finestra grafica a mosaico su MCGA.*

11
Blocchi di bit e animazione

Spostamento di bocchi di bit
CGA e MCGA - EGA e VGA - HGC - Scheda InColor

Operazioni logiche sui pixel
XOR - NOT - AND - OR

Affiancamento di blocchi di bit

Animazione
Animazione con XOR
Sovrapposizione di spostamenti di blocchi di bit

Un cursore in modo grafico
XOR - Spostamento di blocchi di bit

Il presente capitolo tratta lo spostamento di figure all'interno del buffer del video e sullo schermo. Alcuni dei programmi in modalità grafica più utili e divertenti creano l'effetto ottico del movimento su schermo. Oggetti banali come un cursore o insoliti come un'astronave aliena possono simulare il movimento sullo schermo se li cancellate e li ritracciate immediatamente in locazioni diverse. I sistemi di visualizzazione dei PC e dei PS/2 non sono particolarmente adatti per supportare questo tipo di animazione in tempo reale, ma le tecniche trattate in questo capitolo dovrebbero aiutarvi a sfruttare appieno le loro capacità.

Potreste pensare all'animazione su video come se il suo contesto fosse ristretto ai video-game, ma l'animazione trova altri impieghi nella grafica per computer. Ad esempio, tutti i programmi grafici interattivi richiedono un cursore mobile che permette all'utente di puntare le locazioni di schermo. Molti programmi di tracciamento o di progettazione permettono all'utente di spostare forme ed immagini sullo schermo. I programmi di controllo robotico indicano lo stato di un braccio di robot con una rappresentazione animata della sua posizione. Potete creare questo tipo di effetti di animazione utilizzando le tecniche riportate in questo capitolo.

Spostamento di blocchi di bit

Lo strumento software fondamentale per molte tecniche di animazione è rappresentato dallo spostamento di blocchi di bit (una routine che copia un blocco rettangolare di pixel nel/dal/all'interno del buffer del video). La denominazione "spostamento di blocchi di bit" descrive il compito della routine. Dopo tutto, un rettangolo di pixel non è essenzialmente niente di più che un blocco di bit. Tuttavia, una routine per lo spostamento di blocchi di bit può fare molto di più della semplice duplicazione di valori di pixel. Come nel caso di altre routine di tracciamento grafico, una routine per lo spostamento di blocchi di bit può aggiornare i valori di pixel utilizzando le operazioni logiche binarie AND, OR e XOR. Queste operazioni possono creare effetti particolari quando vengono utilizzate come componenti degli spostamenti dei blocchi di bit.

Per copiare un blocco di bit da una locazione ad un'altra all'interno del buffer del video nei sistemi di visualizzazione dei PC e dei PS/2, risulta di solito più efficiente utilizzare un buffer intermedio situato nella RAM di sistema. Per prima cosa si copiano i valori di pixel dal buffer del video al buffer intermedio, quindi si copiano i valori da questo buffer alla posizione desiderata nel buffer del video.

La creazione di una copia intermedia dei pixel in un blocco di bit potrebbe sembrare superflua, ma nella maggior parte delle situazioni è preferibile provare a spostare il blocco di bit interamente all'interno del buffer del video. Ad esempio, né l'EGA né la scheda InColor supportano operazioni logiche dirette (AND, OR e XOR) tra pixel nei piani di bit. Inoltre, gli accessi della CPU alla RAM del video sono più lenti degli accessi equivalenti alla RAM di sistema. Di conseguenza, quando copie multiple dello stesso blocco di bit devono essere memorizzate nel buffer del video, la creazione di una singola copia

nella RAM di sistema e la successiva creazione di copie multiple dalla RAM di sistema alla RAM del video risultano più efficienti.

CGA e MCGA

Il Listato 108 è una routine per lo spostamento di blocchi di bit per la CGA. La routine *GetBitBlock()* copia un blocco di pixel dal buffer del video ad un buffer nella RAM di sistema. La routine complementare *StoreBitBlock()* del Listato 109 copia i pixel dalla RAM di sistema al buffer del video. *StoreBitBlock)(* contiene subroutine per eseguire le operazioni AND, OR o XOR sui pixel nella RAM di sistema utilizzando il contenuto precedente del buffer del video.

```
                TITLE   'Listato 108'
                NAME    GetBitBlock06
                PAGE    55,132

;
; Nome:         GetBitBlock06
;
; Funzione:     Copia un blocco di bit dal buffer del video alla RAM
;                 di sistema nella modalità a 2 colori 640x200
;
; Chiamante:    Microsoft C:
;
;                       int GetBitBlock06(x0,y0,x1,y1,buf);
;
;                       int x0,y0; /* angolo super. sin. del blocco di bit */
;                       int x1,y1;     /* angolo inferiore destro */
;                       char far *buf; /* buffer */
;
; Note:         Ritorna dimensione del blocco di bit nella RAM di sistema.
;


ARGx0           EQU     word ptr [bp+4]
ARGy0           EQU     word ptr [bp+6]
ARGx1           EQU     word ptr [bp+8]
ARGy1           EQU     word ptr [bp+10]
ADDRbuf         EQU              [bp+12]

VARPixelRows    EQU     word ptr [bp-2]
VARPixelRowLenEQU       word ptr [bp-4]
VARincr         EQU     word ptr [bp-6]

ByteOffsetShift         EQU         3   ; rispecchia numero di pixel per byte


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr06:near
```

```
                PUBLIC _GetBitBlock06
_GetBitBlock06PROC   near

                push   bp          ; mantiene registri del chiamante
                mov    bp,sp
                sub    sp,6        ; stabilisce cornice di stack
                push   ds
                push   si
                push   di

; calcola dimensioni del blocco di bit

                mov    ax,ARGx1
                sub    ax,ARGx0
                mov    cx,0FF07h   ; CH := maschera di bit non spostata
                                   ; CL := esegue AND della maschera per AL
                and    cl,al       ; CL := numero di pixel nell'ultimo
                                   ;  byte della riga
                xor    cl,7        ; CL := numero di bit da spostare
                shl    ch,cl       ; CH := maschera di bit per ultimo
                                   ;  byte della riga
                mov    cl,ch
                push   cx          ; salva su stack

                mov    cl,ByteOffsetShift
                shr    ax,cl
                inc    ax          ; AX := numero di byte per riga
                push   ax          ; salva su stack

                mov    ax,ARGy1
                sub    ax,ARGy0
                inc    ax          ; AX := numero di righe di pixel
                push   ax          ; salva su stack

; stabilisce indirizzamento

                mov    ax,ARGy0
                mov    bx,ARGx0
                call   PixelAddr06; ES:BX - >x0,y0 nel buffer del video
                xor    cl,7        ; CL := numero di bit da spostare
                                   ;  a sinistra
                push   es
                pop    ds
                mov    si,bx       ; DS:SI - >buffer del video

                mov    bx,2000h    ; BX := incrementa da prima a seconda
                                   ;  intercalazione nel buffer del video
                                   ;  del CGA
                test   si,2000h
                jz     L01         ; salta se x0,y0 è nella prima
                                   ;  intercalazione

                mov    bx,80-2000h; incrementa da seconda a prima
                                   ;  intercalazione
L01:            mov    VARincr,bx ; inizializza questa variabile
                les    di,ADDRbuf ; ES:DI -> buffer nella RAM di sistema
```

```
; crea intestazione a 5 byte del blocco di bit

            pop     ax
            mov     VARPixelRows,ax
            stosw               ; byte 0-1 := numero di righe di pixel
            pop     ax
            mov     VARPixelRowLen,ax
            stosw               ; byte 2-3 := byte per riga di pixel
            pop     ax
            mov     ch,al       ; CH := maschera di bit per ultimo byte
            stosb               ; byte 4 := maschera di bit per ultimo byte

; copia dal buffer del video alla RAM di sistema

L02:        mov     bx,VARPixelRowLen
            push    si          ; mantiene SI all'inizio della riga di pixel

L03:        lodsw               ; AL := byte successivo nel buffer
                                ;  del video
                                ; AH := (byte successivo) + 1
            dec     si          ; DS:SI -> (byte successivo) + 1
            rol     ax,cl       ; AL := successivi 4 pixel nella riga
            stosb               ; copia sulla RAM di sistema
            dec     bx          ; loop sulla riga
            jnz     L03

            and     es:[di-1],ch    ; maschera ultimo byte di riga
            pop     si          ; DS:SI -> inizio di riga
            add     si,VARincr  ; DS:SI -> inizio di riga successiva
            xor     VARincr,2000h XOR (80-2000H)  ; aggiorna incremento

            dec     VARPixelRows
            jnz     L02         ; loop verso il basso sulle righe

            mov     ax,di
            sub     ax,ADDRbuf  ; AX := ritorna valore (dimens. blocco
                                ;  di bit nella RAM di sistema)

            pop     di          ; ripristina registri ed esce
            pop     si
            pop     ds
            mov     sp,bp
            pop     bp
            ret

_GetBitBlock06ENDP

_TEXT       ENDS

            END
```

Listato 108. *Una routine per copiare un blocco di pixel dal buffer video della CGA alla RAM di sistema.*

```
                TITLE   'Listato  109'
                NAME    StoreBitBlock06
                PAGE    55,132

;
; Nome:         StoreBitBlock06
;
; Funzione:     Copia un blocco di bit dal buffer del video alla RAM
;                di sistema nella modalità a 2 colori 640x200
;
; Chiamante:    Microsoft C:
;
;                       azzera StoreBitBlock06(buf,x,y);
;
;                            char far *buf;   /* buffer */
;                            int x,y;    /* angolo superiore sinistro del
                                                 blocco di bit */
;


ADDRbuf         EQU     dword ptr [bp+4]
ARGx            EQU     word ptr [bp+8]
ARGy            EQU     word ptr [bp+10]

VARPixelRows    EQU     word ptr [bp-2]
VARPixelRowLenEQU       word ptr [bp-4]
VARincr         EQU     word ptr [bp-6]


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

                EXTRN   PixelAddr06:near

                PUBLIC _StoreBitBlock06
_StoreBitBlock06 PROCnear

                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                sub     sp,6        ; stabilisce cornice di stack
                push    ds
                push    si
                push    di

; stabilisce indirizzamento

                mov     ax,ARGy
                mov     bx,ARGx
                call    PixelAddr06; ES:BX -> offset di byte di x,y
                xor     cl,7        ; CL := numero di bit da spostare a destra

                mov     di,bx       ; ES:DI -> x,y nel buffer del video
```

```
            mov     bx,2000h    ; BX := incrementa dalla prima alla seconda
                                ;  intercalazione nel buffer del video
                                ;  del CGA
            test    di,2000h
            jz      L01         ; salta se x,y è nella prima intercalazione

            mov     bx,80-2000h; incrementa da seconda a prima
                                ;  intercalazione

L01:        mov     VARincr,bx ; inizializza questa variabile

            mov     bx,StoreBitBlockOp    ; BX := indirizzo della
                                          ;  subroutine

            lds     si,ADDRbuf ; ES:DI - ;buffer nella RAM del sistema

; ricava dimensioni del blocco di bit dall'intestazione

            lodsw               ; AX := numero di righe di pixel
            mov     VARPixelRows,ax
            lodsw               ; AX := byte per riga di pixel
            mov     VARPixelRowLen,ax
            lodsb               ; AL := maschera di bit per ultimo byte
                                ;  nella riga
            mov     ch,al
            jmp     bx          ; salta a subroutine
ReplaceBitBlock:
            cmp     cx,0FF00h   ; se maschera<> 0FFH o bit da spostare<> 0
            jne     L15         ;  salta se non allineato al byte

; routine per blocchi di bit allineati al byte

            mov     cx,VARPixelRowLen

L10:        push    di          ; mantiene DI e CX
            push    cx
            rep     movsb       ; copia una riga di pixel nel buffer
                                ;  del video
            pop     cx          ; ripristina DI e CX
            pop     di
            add     di,VARincr  ; ES:DI -> successiva riga di pixel
                                ;  nel buffer
            xor     VARincr,2000h XOR (80-2000h)  ; aggiorna incremento
            dec     VARPixelRows
            jnz     L10         ; loop verso il basso su righe di pixel

            jmp     Lexit

; routine per tutti gli altri blocchi di bit

L15:        not     ch          ; CH := maschera per fine riga
            mov     dx,0FF00h
            ror     dx,cl       ; DX := maschera ruotata per ogni byte
            mov     bx,VARPixelRowLen
            dec     bx          ; BX := byte per riga - 1
```

```
L16:        push    di
            test    bx,bx
            jz      L18         ; salta se solo un byte per riga

            push    bx

L17:        and     es:[di],dx  ; maschera successivi 8 pixel nel buffer
                                ;  del video
            lodsb               ; AL := pixel nel blocco di bit
            xor     ah,ah
            ror     ax,cl       ; AX := pixel ruotati in posizione
            or      es:[di],ax  ; imposta pixel nel buffer del video
            inc     di          ; ES:DI -> byte successivo
            dec     bx          ;  nel blocco di bit
            jnz     L17

            pop     bx

L18:        mov     al,ch
            mov     ah,0FFh     ; AX := maschera per ultimi pixel
                                ;  nella riga
            ror     ax,cl       ; AX := maschera ruotata in posizione
            and     es:[di],ax  ; maschera ultimi pixel nel buffer
                                ;  del video
            lodsb               ; AL := ultimo byte nella riga
            xor     ah,ah
            ror     ax,cl       ; AX := pixel ruotati in posizione
            or      es:[di],ax  ; imposta pixel nel buffer del video

            pop     di
            add     di,VARincr  ; ES:DI -> successiva riga di pixel
                                ;  nel buffer
            xor     VARincr,2000h XOR (80-2000h)
            dec     VARPixelRows
            jnz     L16         ; loop verso il basso su righe di pixel

            jmp     Lexit
XORBitBlock:
            mov     bx,VARPixelRowLen

L20:        push    di
            push    bx

L21:        lodsb               ; AL := pixel nel blocco di bit
            xor     ah,ah
            ror     ax,cl       ; AX := pixel ruotati in posizione
            xor     es:[di],ax  ; esegue XOR dei pixel
                                ;  nel buffer del video
            inc     di          ; ES:DI -> byte successivo
                                ;  nel blocco di bit
            dec     bx
            jnz     L21
            pop     bx
            pop     di
            add     di,VARincr  ; ES:DI -> successiva riga di pixel
                                ;  nel buffer
```

```
            xor     VARincr,2000h XOR (80-2000h)
            dec     VARPixelRows
            jnz     L20         ; loop verso il basso su righe di pixel

            jmp     Lexit

ANDBitBlock:
            not     ch          ; CH := maschera per fine riga

            mov     bx,VARPixelRowLen
            dec     bx          ; BX := byte per riga - 1

L30:        push    di
            test    bx,bx
            jz      L32         ; salta se solo un byte per riga

            push    bx

L31:        lodsb               ; AL := pixel nel blocco di bit
            mov     ah,0FFh
            ror     ax,cl       ; AX := pixel ruotati in posizione
            and     es:[di],ax  ; esegue AND dei pixel nel buffer
                                ;  del video
            inc     di          ; ES:DI -> byte successivo
                                ;  nel blocco di bit
            dec     bx
            jnz     L31

            pop     bx

L32:        lodsb               ; AL := ultimo byte nella riga
            or      al,ch       ; maschera ultimi pixel nella riga
            mov     ah,0FFh
            ror     ax,cl       ; AX := pixel ruotati in posizione
            and     es:[di],ax  ; esegue AND dei pixel nel buffer
                                ;  del video

            pop     di
            add     di,VARincr  ; ES:DI -> successiva riga di pixel
                                ;  nel buffer
            xor     VARincr,2000h XOR (80-2000h)
            dec     VARPixelRows
            jnz     L30         ; loop verso il basso su righe di pixel
            jmp     Lexit

RBitBlock:
            mov     bx,VARPixelRowLen

L40:        push    di
            push    bx

L41:        lodsb               ; AL := pixel nel blocco di bit
            xor     ah,ah
            ror     ax,cl       ; AX := pixel ruotati in posizione
            or      es:[di],ax  ; esegue OR dei pixel nel buffer del video
            inc     di          ; ES:DI -> byte successivo nel blocco di bit
```

```
            dec     bx
            jnz     L41

            pop     bx
            pop     di
            add     di,VARincr ; ES:DI -> successiva riga di pixel nel buffer
            xor     VARincr,2000h XOR (80-2000h)
            dec     VARPixelRows

            jnz     L40        ; loop verso il basso su righe di pixel


Lexit:      pop     di         ; ripristina registri ed esce
            pop     si
            pop     ds
            mov     sp,bp
            pop     bp
            ret

_StoreBitBlock06 ENDP

_TEXT       ENDS


_DATA       SEGMENTword public 'DATA'

StoreBitBlockOp      DW    ReplaceBitBlock; indirizzo della subroutine
                                          ; selezionata(Sostituzione,
                                          ; XOR, AND, OR)

_DATA       ENDS

            END
```

Listato 109. *Una routine per copiare un bocco di pixel dalla RAM di sistema al buffer video del CGA.*

Nella modalità a 2 colori 640 per 480 e nella modalità a 256 colori 320 per 200 dell'MCGA, l'indirizzamento dei pixel è diverso rispetto a quello delle due modalità compatibili CGA. Per il resto, le versioni di *GetBitBlock()* e di *StoreBitBlock()* sono simili in tutte le modalità MCGA.

EGA e VGA

Nelle modalità grafiche EGA e VGA originali, la routine di spostamento dei blocchi di bit deve spostare il contenuto di tutti i quattro piani di bit alla RAM di sistema. La routine *GetBitBlock()* del Listato 110 estrae i byte da ogni piano di bit utilizzando la modalità di lettura 0 e selezionando ogni piano di bit in successione con il registro di lettura della maschera di mappa del controller grafico. *StoreBitBlock()*, nel Listato 111, utilizza quindi la modalità di scrittura 0 per copiare i dati nei piani di bit. I piani di bit vengono

isolati nella modalità di scrittura 0 impostando opportunamente il registro di maschera di mappa del sequencer.

Non utilizzate le routine dei Listati 110 e 111 su EGA con solo 64 KB di RAM video. Dal momento che le mappe di memoria sono concatenate tra loro per formare i due piani di bit usati nelle modalità grafiche 640 per 350, queste routine non funzioneranno correttamente in questa situazione (il Capitolo 4 tratta questo argomento più approfonditamente).

```
                TITLE   'Listato 110'
                NAME    GetBitBlock10
                PAGE    55,132

;
; Nome:         GetBitBlock10
;
; Funzione:     Copia un blocco di bit dal buffer del video alla RAM
;                di sistema nelle modalità grafiche EGA e VGA originali
;
; Chiamante:    Microsoft C:
;
;                       int GetBitBlock10(x0,y0,x1,y1,buf);
;
;                            int x0,y0;  /* angolo superiore sinistro del
                                            blocco di bit */
;                            int x1,y1;      /* angolo inferiore destro */
;                            char far *buf;  /* buffer */
;
; Note:         Ritorna dimensione del blocco di bit nella RAM di sistema.
;


ARGx0           EQU     word ptr [bp+4]
ARGy0           EQU     word ptr [bp+6]
ARGx1           EQU     word ptr [bp+8]
ARGy1           EQU     word ptr [bp+10]
ADDRbuf         EQU              [bp+12]

VARPixelRows    EQU     word ptr [bp-2]
VARPixelRowLenEQU       word ptr [bp-4]

BytesPerRow     EQU     80
ByteOffsetShift EQU     3           ; rispecchia numero di pixel per byte


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr10:near

                PUBLIC _GetBitBlock10
_GetBitBlock10PROC      near

                push    bp          ; mantiene registri del chiamante
```

```
            mov     bp,sp
            sub     sp,4        ; stabilisce cornice di stack
            push    ds
            push    si
            push    di

; calcola dimensioni del blocco di bit

            mov     ax,ARGx1
            sub     ax,ARGx0
            mov     cx,0FF07h   ; CH := maschera di bit non spostata
                                ; CL := esegue AND della maschera per AL
            and     cl,al       ; CL := numero di pixel nell'ultimo
                                ;  byte della riga
            xor     cl,7        ; CL := numero di bit da spostare
            shl     ch,cl       ; CH := maschera di bit per ultimo byte
                                ;  della riga
            mov     cl,ch
            push    cx          ; salva su stack

            mov     cl,ByteOffsetShift
            shr     ax,cl
            inc     ax          ; AX := numero di byte per riga
            push    ax          ; salva su stack

            mov     ax,ARGy1
            sub     ax,ARGy0
            inc     ax          ; AX := numero di righe di pixel
            push    ax          ; salva su stack

; stabilisce indirizzamento

            mov     ax,ARGy0
            mov     bx,ARGx0
            call    PixelAddr10; ES:BX -> x0,y0 nel buffer del video
            xor     cl,7        ; CL := numero di bit da spostare a sin.
            push    es
            pop     ds
            mov     si,bx       ; DS:SI -> buffer del video

            les     di,ADDRbuf  ; ES:DI -> buffer nella RAM di sistema

; crea intestazione a 5 byte del blocco di bit

            pop     ax
            mov     VARPixelRows,ax
            stosw               ; byte 0-1 := numero di righe di pixel
            pop     ax
            mov     VARPixelRowLen,ax
            stosw               ; byte 2-3 := byte per riga di pixel
            pop     ax
            mov     ch,al       ; CH := maschera di bit per ultimo byte
                                ;  nella riga
            stosb               ; byte 4 := maschera di bit per
                                ;  ultimo byte
```

```
; imposta controller grafico
              mov    dx,3CEh     ; DX := porta indirizzo controller grafico

              mov    ax,0005     ; AH := 0 (modalità di lettura 0,
                                 ;  modalità di scrittura 0)
                                 ; AL := 5 (numero registro modalità)
              out    dx,ax       ; imposta modalità di lettura 0

              mov    ax,0304h    ; AH := 3 (primo piano di bit da leggere)
                                 ; AL := 4 (numero reg selez. mappa
                                 ;  di lettura)

; copia dal buffer del video alla RAM di sistema

L01:          out    dx,ax       ; seleziona succ. mappa di memoria
                                 ;  da leggere
              push   ax          ; mantiene numero mappa memoria
              push   VARPixelRows    ; mantiene numero di righe di pixel
              push   si          ; mantiene offset di x0,y0

L02:          mov    bx,VARPixelRowLen
              push   si          ; mantiene SI all'inizio della riga
                                 ;  di pixel

L03:          lodsw              ; AL := byte successivo nel buffer
                                 ;  del video
                                 ; AH := (byte successivo) + 1
              dec    si          ; DS:SI -> (byte successivo) + 1
              rol    ax,cl       ; AL := successivi 4 pixel nella riga
              stosb              ; copia sulla RAM di sistema
              dec    bx          ; loop sulla riga
              jnz    L03

              and    es:[di-1],ch    ; maschera ultimo byte di riga
              pop    si              ; DS:SI -> inizio di riga
              add    si,BytesPerRow  ; DS:SI -> inizio di riga successiva

              dec    VARPixelRows
              jnz    L02         ; loop verso il basso sulle righe

              pop    si          ; DS:SI -> inizio blocco di bit
              pop    VARPixelRows    ; ripristina numero di righe di pixel
              pop    ax          ; AH := ultima lettura di mappa
                                 ; AL := numero reg selezione mappa lettura
              dec    ah
              jns    L01         ; loop su piani di bit

              mov    ax,di
              sub    ax,ADDRbuf  ; AX := ritorna valore (dimens. blocco
                                 ;  di bit nella RAM di sistema)

              pop    di          ; ripristina registri ed esce
              pop    si
              pop    ds
              mov    sp,bp

              pop    bp
              ret
```

```
_GetBitBlock10ENDP

_TEXT           ENDS

                END
```

Listato 110. *Una routine per copiare un blocco di pixel dal buffer del video dell'EGA o della VGA alla RAM di sistema nelle modalità grafiche originali.*

```
                TITLE   'Listato  111'
                NAME    StoreBitBlock10
                PAGE    55,132

;
; Nome:         StoreBitBlock10
;
; Funzione:     Copia un blocco di bit dal buffer del video alla RAM
;               di sistema nelle modalità grafiche EGA e VGA originali
;
; Chiamante:    Microsoft C:
;
;                       azzera StoreBitBlock10(buf,x,y);
;
;                            char far *buf;   /* buffer */
;                            int x,y;    /* angolo superiore sinistro del
                                                blocco di bit */
;


ADDRbuf                 EQU         dword ptr [bp+4]
ARGx                    EQU         word ptr [bp+8]
ARGy                    EQU         word ptr [bp+10]

VARPixelRows            EQU         word ptr [bp-2]
VARPixelRowLen          EQU         word ptr [bp-4]
VARRowCounter           EQU         word ptr [bp-6]
VARStartMask            EQU         word ptr [bp-8]
VAREndMaskL             EQU         word ptr [bp-10]
VAREndMaskR             EQU         word ptr [bp-12]

BytesPerRow             EQU         80  ; larghezza logica del buffer
                                        ;  del video
ByteOffsetShift         EQU         3   ; rispecchia numero di pixel per byte
RMWbits                 EQU         18h ; seleziona sostituzione, XOR,
                                        ;  AND, o OR

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr10:near

                PUBLIC _StoreBitBlock10
_StoreBitBlock10        PROC        near
```

```
            push    bp          ; mantiene registri del chiamante
            mov     bp,sp
            sub     sp,12       ; stabilisce cornice di stack
            push    ds
            push    si
            push    di

; stabilisce indirizzamento

            mov     ax,ARGy
            mov     bx,ARGx
            call    PixelAddr10; ES:BX -> offset di byte di x,y
            inc     cl
            and     cl,7        ; CL := numero di bit da spostare a sinistra

            mov     di,bx       ; ES:DI -> x,y nel buffer del video

            lds     si,ADDRbuf  ; ES:DI -> buffer nella RAM del sistema

; ricava dimensioni del blocco di bit dall'intestazione

            lodsw               ; AX := numero di righe di pixel
            mov     VARPixelRows,ax
            lodsw               ; AX := byte per riga di pixel
            mov     VARPixelRowLen,ax
            lodsb               ; AL := maschera di bit per ultimo byte
                                ;  nella riga
            mov     ch,al

; imposta controller grafico

            mov     dx,3CEh     ; DX := porta di I/O del controller grafico

            mov     ah,RMWbits  ; AH := valore per registro rotazione
            mov     al,3        ;  dati/selezione funzione
            out     dx,ax       ; aggiorna questo registro

            mov     ax,0805h    ; AH := 8 (modalità lettura 1, modalità
                                ;  scrittura 0)
                                ; AL := 5 (numero registro modalità)
            out     dx,ax       ; imposta modalità lettura 0

            mov     ax,0007     ; AH := 0 (ininfluente per tutte le mappe;
                                ;  letture di CPU ritornano sempre 0FFH)
                                ; AL := 7 (numero reg colore ininfluente)
            out     dx,ax       ; imposta reg colore ininfluente

            mov     ax,0FF08h   ; AH := 0FFH (valore per reg maschera
                                ;  di bit)
            out     dx,ax       ; imposta reg maschera di bit
            mov     dl,0C4h     ; DX := 3C4H (porta di I/O del sequencer)
            mov     ax,0802h    ; AH := 1000B (valore per reg maschera di
                                ;  mappa)
                                ; AL := 2 (numero registro maschera di mappa)
            cmp     cx,0FF00h   ; se maschera <> OFFH o bit da spostare<> 0
            jne     L15         ;  salta se non allineato al byte
```

```
; routine per blocchi di bit allineati al byte

            mov     cx,VARPixelRowLen

L10:        out     dx,ax       ; abilita un piano di bit per scritture
            push    ax          ; mantiene valore di maschera di mappa
            push    di          ; mantiene offset di buffer del video di x,y
            mov     bx,VARPixelRows


11:         push    di          ; mantiene DI e CX
            push    cx

L12:        lodsb               ; AL := successivo byte di pixel
            and     es:[di],al  ; aggiorna piano di bit
            inc     di
            loop    L12

            pop     cx          ; ripristina DI e CX
            pop     di
            add     di,BytesPerRow ; ES:DI -> successiva riga di pixel
                                   ;  nel buffer
            dec     bx
            jnz     L11         ; loop verso il basso su righe di pixel

            pop     di          ; ES:DI -> offset del buffer del video di x,y
            pop     ax          ; AH := valore corrente reg maschera
                                ;  di mappa
            shr     ah,1        ; AH := nuovo valore di maschera di mappa
            jnz     L10         ; loop su tutti i piani di bit

            jmp     Lexit

; routine per blocchi di bit non allineati

L15:        push    ax          ; mantiene valori reg maschera di mappa

            mov     bx,0FFh     ; BH := 0 (maschera per primo byte
                                ;  nella riga)
                                ; BL := 0FFh
            mov     al,ch       ; AL := maschera per ultimo byte nella
                                ;  riga di pixel
            cbw                 ; AH := 0FFh (maschera per ultimo byte -1)

            cmp     VARPixelRowLen,1
            jne     L16         ; salta se più di un byte per riga

            mov     bl,ch
            mov     ah,ch       ; AH := maschera per ultimo byte -1
            xor     al,al       ; AL := 0 (maschera per ultimo byte)


L16:        shl     ax,cl       ; sposta maschere in posizione
            shl     bx,cl
            mov     bl,al       ; salva maschere insieme a ..
            mov     al,8        ; .. numero registro maschera di bit
            mov     VAREndMaskL,ax
            mov     ah,bl
```

```
            mov     VAREndMaskR,ax
            mov     ah,bh
            mov     VARStartMask,ax

            mov     bx,VARPixelRowLen
            pop     ax          ; ripristina valori reg maschera di mappa

; imposta pixel riga per riga nei piani di bit
L17:        out     dx,ax       ; abilita un piano di bit per scritture
            push    ax          ; mantiene valore di maschera di mappa
            push    di          ; mantiene offset del buffer del video
                                ;  di x,y
            mov     dl,0CEh     ; DX := 3CEH (porta controller grafico)

            mov     ax,VARPixelRows
            mov     VARRowCounter,ax     ; inizializza contatore di loop

; imposta pixel all'inizio di riga nel piano di bit correntemente
;abilitato

L18:        push    di          ; mantiene offset dell'inizio
                                ;  della riga di pixel
            push    si          ; mantiene offset della riga nel blocco
                                ;  di bit
            push    bx          ; mantiene byte per riga di pixel

            mov     ax,VARStartMask
            out     dx,ax       ; imposta reg maschera di bit
                                ;  per primo byte della riga

            lodsw               ; AH := secondo byte di pixel
                                ; AL := primo byte di pixel
            dec     si          ; DS:SI -> secondo byte di pixel
            test    cl,cl
            jnz     L19         ; salta se non allineato a sinistra

            dec     bx          ; BX := byte per riga - 1
            jnz     L20         ; salta se almeno 2 byte per riga
            jmp     short L22   ; salta se solo un byte per riga

L19:        rol     ax,cl       ; AH := parte sinistra del primo byte,
                                ;       parte destra del secondo byte
                                ; AL := parte destra del primo byte,
                                ;       parte sinistra del secondo byte
            and     es:[di],ah  ; imposta pixel per parte sinistra
                                ;  primo byte
            inc     di
            dec     bx          ; BX := byte per riga - 2
L20:        push    ax          ; mantiene pixel
            mov     ax,0FF08h
            out     dx,ax       ; imposta reg maschera di bit per byte succ.
            pop     ax

            dec     bx
            jng     L22         ; salta se solo 1 o 2 byte nella riga
                                ;  di pixel
```

```
; imposta pixel nel centro della riga

L21:            and     es:[di],al ; imposta pixel nella parte destra del byte
                                   ;  corrente e nella parte sinistra
                inc     di         ;  del byte successivo

                lodsw              ; AH := successivo byte +1 di pixel
                dec     si         ; AL := byte successivo di pixel
                rol     ax,cl      ; AH := parte sinistra del byte successivo,
                                   ;       parte destra del byte successivo +1

                                   ; AL := parte destra del byte successivo,
                                   ;       parte sinistra del byte successivo +1
                dec     bx
                jnz     L21        ; loop su righe di pixel

; imposta pixel alla fine della riga

L22:            mov     bx,ax      ; BH := parte destra dell'ultimo byte,
                                   ;       parte dell'ultimo byte -1
                                   ; BL := parte sinistra dell'ultimo byte,
                                   ;       parte destra dell'ultimo byte -1
                mov     ax,VAREndMaskL ; AH := maschera per ultimo byte -1
                                   ; AL := numero reg maschera di bit
                out     dx,ax      ; imposta registro maschera di bit
                and     es:[di],bl ; imposta pixel per ultimo byte -1

                mov     ax,VAREndMaskR ; maschera per ultimo byte nella riga
                                       ;  di pixel
                out     dx,ax      ; .. ultimo byte nella riga di pixel
                and     es:[di+1],bh   ; imposta pixel per ultimo byte

                pop     bx         ; BX := byte per riga di pixel
                pop     si
                add     si,bx      ; DS:SI -> riga successiva nel blocco
                                   ;  di bit
                pop     di
                add     di,BytesPerRow ; ES:DI -> successiva riga di pixel
                                       ;  nel buffer
                dec     VARRowCounter
                jnz     L18        ; loop verso il basso su righe di pixel

                pop     di         ; ES:DI -> offset del buffer del video di x,y
                pop     ax         ; AX := valore corrente maschera di mappa
                mov     dl,0C4h    ; DX := 3C4H
                shr     ah,1       ; AH := valore successivo maschera di mappa
                jnz     L17        ; loop su piani di bit
; riporta controller grafico e sequencer a loro stati di default

Lexit:          mov     ax,0F02h   ; valore di default di maschera di mappa
                out     dx,ax

                mov     dl,0CEh    ; DX := 3CEh
                mov     ax,0003    ; rotazione dati/selezione funzione
                                   ;  di default
```

```
                out     dx,ax
                mov     ax,0005     ; valore di modalità di default
                out     dx,ax
                mov     ax,0F07h    ; valore di confronto colore di default
                out     dx,ax

                mov     ax,0FF08h   ; valore di maschera di bit di default
                out     dx,ax

                pop     di          ; ripristina registri ed esce
                pop     si
                pop     ds
                mov     sp,bp
                pop     bp
                ret

_StoreBitBlock10 ENDP

_TEXT           ENDS
                END
```

Listato 111. *Una routine per copiare un blocco di pixel dalla RAM di sistema al buffer del video dell'EGA o della VGA nelle modalità grafiche originali.*

HGC

Le routine per lo spostamento di blocchi di bit per la modalità grafica monocromatica 720 per 348 per HGC e HGC+ sono analoghe a quelle per la modalità a 2 colori 640 per 200 del CGA. Le differenze risiedono in come vengono calcolati gli indirizzi di pixel e nel modo in cui il buffer del video viene intercalato.

Scheda InColor

le routine per la modalità a 16 colori 720 per 348 della scheda InColor assomigliano alle routine per EGA dei Listati 110 e 111, in quanto i buffer del video di entrambi gli adattatori vengono mappati in piani di bit paralleli. Le differenze tra le routine risiedono nel modo in cui vengono calcolati gli indirizzi di pixel, in come viene intercalato il buffer del video e nel modo in cui si accede ai piani di bit individuali. Su scheda InColor, potete utilizzare la stessa tecnica di *ReadPixelInC()* (trattata nel Capitolo 5) per programmare i registri di colore e controllo lettura/scrittura ed isolare il contenuto di ogni piano di bit. Analogamente, una routine per la memorizzazione di blocchi di bit per la scheda InColor segue *StorePixelInC()* per quanto riguarda l'uso del registro di maschera di piano e i registri di colore e controllo lettura/scrittura.

Operazioni logiche sui pixel

Se avete sperimentato gli esempi di programmazione dei pixel e di tracciamento linee dei capitoli precedenti, probabilmente saprete perché le operazioni logiche binarie (XOR, AND e OR) sono utili nella programmazione grafica del video. In questo caso, potete saltare i prossimi paragrafi. In caso contrario, continuate la lettura per scoprire come i programmi grafici di visualizzazione possono sfruttare la capacità di eseguire operazioni-XOR, AND e OR sui valori di pixel.

XOR

L'operazione XOR è utile in quanto è reversibile. Quando modificate il valore di un pixel nel buffer del video utilizzando la funzione XOR, potete riportare il pixel al valore originale ripetendo l'operazione. Ad esempio, se un pixel del buffer del video ha il valore 9, l'impostazione del suo valore tramite l'XOR con un valore 5 fornisce come risultato un valore di pixel 0CH. Se si riesegue l'XOR del valore di pixel risultante (0CH) con il valore 5, si riporta il pixel al valore originario di 9.

Ciò implica che potete effettuare l'XOR di oggetti nel buffer del video ed in direzione dello schermo, e poi cancellarli, senza preoccuparvi di salvarli e di ripristinare il contenuto del buffer del video. L'uso di XOR ha comunque dei limiti. Uno di questi è che un'immagine contenente pixel con valore zero non può essere sottoposta a XOR all'interno del buffer del video. Dal momento che effettuare l'XOR di un pixel con 0 lascia invariato il valore del pixel, solo i pixel diversi da zero dell'immagine influenzeranno il buffer del video.

Un altro limite più serio è che uno sfondo composto da un modello grafico può oscurare l'immagine che state tentando di sottoporre a XOR all'interno del buffer del video. Osservate la Figura 114, nella quale una stringa di testo viene sottoposta a XOR con sfondi progressivamente confusi. Il testo è perfettamente leggibile su sfondo pieno, ma uno sfondo a strisce confonde significativamente le lettere. Nel caso peggiore, effettuare l'XOR di un'immagine monocolore con un modello di pixel casuali può produrre come risultato un altro modello di pixel casuali.

NOT

Un'operazione logica NOT su un valore di pixel commuta a 0 tutti i bit 1 e a 1 tutti i bit 0. Ovviamente, due operazioni NOT in sequenza lasceranno immutato il valore del pixel. Una comune pratica di programmazione nelle modalità grafiche monocromatiche è quella di utilizare NOT per commutare uno stato di inversione video. Ad esempio, un carattere nero su bianco può essere invertito effettuando le operazioni NOT sui suoi pixel.

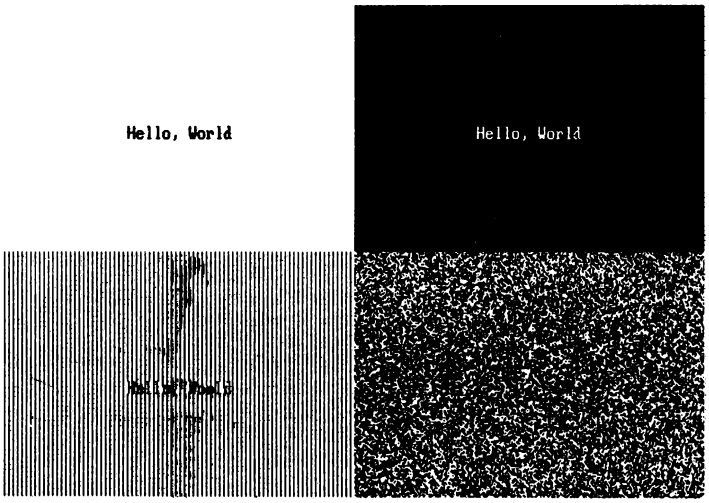


Figura 114. *Effetti di XOR su una stringa di testo con vari sfondi.*

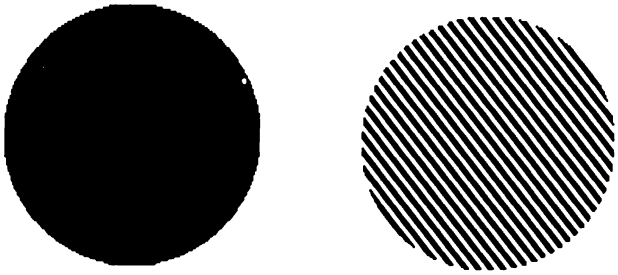
L'effetto di NOT su valori di pixel multibit risulta meno chiaro. In questa situazione, l'operazione NOT trasforma un valore di pixel in un altro valore di pixel, ma i colori corrispondenti a questi due valori potrebbero non essere correlati. Di conseguenza, in una modalità grafica a colori, l'esecuzione di un'operazione NOT su tutti i pixel di una matrice di carattere modifica i valori sia di sfondo sia di primo piano, ma la combinazione di colore risultante potrebbe essere particolarmente sgradevole o addirittura illeggibile. Nella manipolazione dei pixel nella grafica a colori, utilizzate con cautela NOT.

CONSIGLIO

L'esecuzione di un'operazione logica NOT è equivalente all'esecuzione di un'operazione logica XOR utilizzando un valore binario di tutti i bit 1. Questo significa che potete utilizzare qualsiasi routine XOR per i pixel sviluppata in questo libro anche per eseguire operazioni NOT. Di conseguenza, la scrittura di routine apposite NOT non fornirebbe grandi vantaggi nella manipolazione dei pixel.

AND

Anche l'operazione logica binaria AND si rivela utile nella manipolazione delle immagini grafiche. Considerate, ad esempio, come potreste faticare per tracciare il cerchio a strisce della Figura 115b. Potreste realizzarlo nel modo più faticoso, intersecando una serie di linee parallele con il cerchio. Questa procedura sarebbe laboriosa, però, a causa della programmazione supplementare e dei calcoli necessari per la determinazione dei punti di intersezione.



a. b.

Figura 115. *Uso di AND per tracciare un cerchio a strisce. Il cerchio nella Figura 115a è costituito da pixel con il valore massimo possibile. Le linee vengono tracciate sulla superficie del cerchio utilizzando un' operazione AND sui pixel per produrre il cerchio a strisce della Figura 115b.*

Il metodo più semplice consiste nel tracciare un cerchio pieno (Figura 115a) assegnando ai pixel il valore massimo possibile (cioè, tutti i bit impostati a 1) su uno sfondo di pixel con valore zero; questo cerchio verrà poi utilizzato come maschera con la quale effettuare l'AND dei pixel nelle linee parallele. Quando i pixel di ogni linea vengono sottoposti a AND con i pixel all'interno del cerchio, i loro valori originali vengono memorizzati immutati nel buffer del video. Al di fuori del cerchio, il risultato dell'operazione AND sui pixel di linea con lo sfondo zero fornisce sempre come risultato la memorizzazione nel buffer di pixel con valore zero. Ne risulta così un cerchio a strisce.

Potete applicare questa tecnica a qualsiasi forma grafica, ma risulta particolarmente attraente in combinazione ad una routine di spostamento blocchi di bit. Potete sovrapporre immagini costituite da modelli grafici con una breve sequenza di spostamenti di blocchi di bit utilizzando operazioni AND e OR sui pixel. Nella Figura 116, un'area circolare del modello grafico B viene sovrapposta al modello grafico A utilizzando una maschera per isolare un "foro" nel modello A. Il contrario della stessa maschera estrae la parte piena del modello B. I due modelli grafici mascherati vengono quindi sovrapposti da un terzo spostamento di blocco di bit che usa OR (o XOR) per aggiornare i pixel.

OR

L'operatore logico OR viene utilizzato più raramente di XOR per la manipolazione dei valori di pixel. L'operazione OR, a differenza di XOR o di NOT, non è reversibile. Il risultato di OR sui pixel dipende sempre dai loro valori precedenti contenuti nel buffer del video.

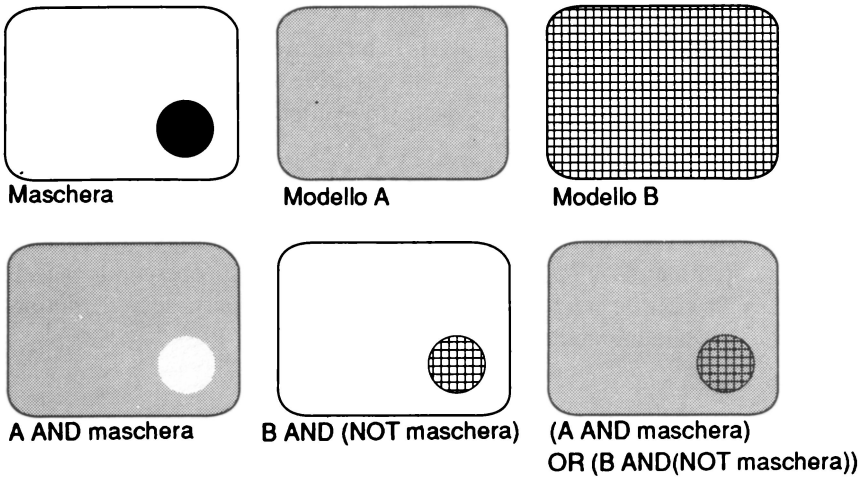


Figura 116. *Mascheramento di immagini a modelli grafici con operazioni AND sui pixel*

Un impiego tipico dell'operazione OR sui pixel è quello che produce l'accentuazione dei punti di intersezione di forme all'interno del buffer del video. Considerate cosa accadrebbe se effettuaste l'OR di due aree di diverso colore nel buffer del video a 16 colori (vedere Figura 117). Se un rettangolo viene riempito con pixel di valore 3 e l'altro rettangolo con pixel di valore 5, i pixel dei punti di intersezione avranno il valore 7 (3 OR 5). Con la normale tavolozza di colore di default, il rettangolo superiore appare ciano, il rettangolo inferiore viola e l'intersezione è bianca.

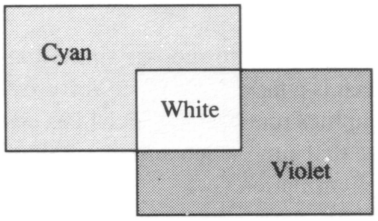


Figura 117. *Esecuzione di OR su due aree colorate nel buffer del video a 16 bit.*

Affiancamento di blocchi di bit

Potete utilizzare le routine di spostamento blocchi di bit per riempire un'area del buffer del video con un qualsiasi modello grafico arbitrario. Questa operazione viene realizzata riempiendo il buffer con spostamenti di blocchi di bit per aree rettangolari del buffer (vedere Figura 118). L'uso della tecnica di sottoporre i valori ad un AND con una maschera permette di riempire una qualsiasi forma arbitraria, come ad esempio il cerchio della Figura 119, con un modello grafico contenuto in un blocco di bit.

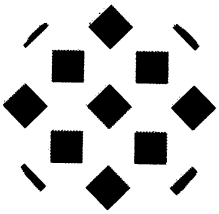


Figura 118. *Affiancamento di blocchi di bit.*

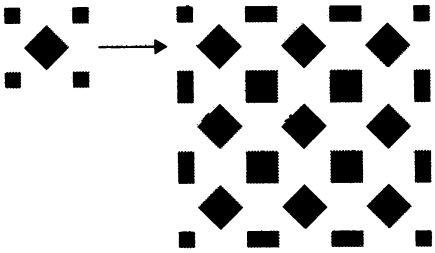


Figura 119. *Affiancamento tramite AND con una maschera.*

CONSIGLIO

Potete usare una variante dell'affiancamento di blocchi di bit come una sorta di generatore di caratteri software. Se definite un gruppo di blocchi di bit, ognuno dei quali rappresenta un carattere di un set di caratteri, potete coprire lo schermo con i caratteri utilizzando l'affiancamento. Questa è una tecnica per la visualizzazione di caratteri spaziati in modo proporzionale.

Animazione

I sottosistemi di visualizzazione dei PC e dei PS/2 non hanno hardware incorporato per supportare l'animazione. Di conseguenza, lo spostamento delle immagini sullo schermo è un compito relegato al software (ciò rappresenta un buon motivo per rendere le vostre routine grafiche le più efficienti possibili). Diverse tecniche software possono produrre animazione video in tempo reale. Ogni tecnica è più adatta ad un particolare tipo di animazione.

Animazione con XOR

Potete sfruttare la reversibilità dell'operazione logica XOR per simulare il movimento sullo schermo di qualsiasi pixel o serie di pixel. Per simulare il movimento di un oggetto, sottoponete due volte l'oggetto a XOR all'interno del buffer del video. L'oggetto lampeggia sullo schermo la prima volta che viene tracciato. Successivamente scompare immediatamente la seconda volta che viene tracciato. Se ritracciate ripetutamente l'oggetto in una posizione leggermente diversa, esso sembrerà spostarsi sullo schermo.

Contorni

Considerate il frammento di programma in C del Listato 112. Questa parte di codice simula l'espansione di un cerchio a partire dal suo centro effettuando l'XOR ripetutamente del cerchio all'interno del buffer del video con un raggio gradualmente crescente.

```
/* Listato 112 */

main()
{
              int    xc          = 400;     /* centro del cerchio */
              int    yc          = 125;
              int    a,b;                   /* assi semimaggiore */
                                            /*  e semiminore */
              int    n = 12;                /* valore di pixel */
              int    i;
              float  ScaleFactor = 1.37;    /* per modalità a 16 */
                                            /*  colori 640x350 */
              for( i=0; i<10; i++ )
                for( a=0; a<100; a++ )
                {
                  b = (float) a / ScaleFactor;
                                    /* scala asse semiminore */
                  Ellipse10( xc, yc, a, b, n );
                                    /* traccia un cerchio */
                  Ellipse10( xc, yc, a, b, n );   /* lo ritraccia */
                }
}
```

Listato 112. *Esecuzione di XOR su un cerchio all'interno del buffer del video.*

Questa tecnica viene utilizzata frequentemente in modo interattivo per tracciare il contorno di un'area rettangolare dello schermo. Il contorno viene rapidamente sottoposto a XOR all'interno e all'esterno del buffer del video mentre l'utente sposta un dispositivo di puntamento come ad esempio un mouse. Esattamente come il cerchio creato dalla routine del Listato 112 simula un'espansione, un contorno rettangolare può simulare uno spostamento, un'espansione o un restringimento in risposta alle azioni dell'utente.

La routine del Listato 113 fa scorrere un rettangolo sullo schermo. Ad ogni iterazione, il rettangolo viene tracciato e quindi cancellato utilizzando linee che vengono sottoposte a XOR all'interno del buffer del video. In questo esempio, la locazione su schermo del rettangolo viene modificata all'interno di un loop iterativo. In pratica, però, la dimensione e la locazione del rettangolo potrebbero essere modificate in risposta ad un input da tastiera o da un dispositivo di puntamento. In questo caso, il rettangolo verrebbe cancellato e ritracciato ogni volta che l'input indica una modifica della posizione.

```
/* Listato 113 */

#define Xmax          640

#define        TRUE   1
#define        FALSE  0

main()
{

               int    x0          = 0;            /* angoli del riquadro */
                                                  /* a 0,0 e 150,100 */
               int    y0          = 0;
               int    x1          = 150;
               int    y1          = 100;
               int    n           = 12;           /* valore di pixel */


                      while( x1 < Xmax )          /* fa scorrere riquadro */
                                                  /*  verso destra */
                        XORBox(x0++,y0,x1++,y1,n);

                      while( x0 > 0 )             /* fa scorrere riquadro */
                                                  /*  verso sinistra */
                        XORBox(--x0,y0,--x1,y1,n);
}
XORBox                ( x0, y0, x1, y1, n )
int                   x0,y0,x1,y1;                /* coordinate di pixel */
                                                  /*  degli angoli opposti */
int                   n;                          /* valore di pixel */
{
                      Rectangle(x0,y0,x1,y1,n);   /* traccia il riquadro */
                      Rectangle(x0,y0,x1,y1,n ;   /* cancella il riquadro */
}


Rectangle( x0, y0, x1, y1, n )
int                   x0,y0,x1,y1;
```

```
int                     n;
{
                        Line10( x0, y0, x0, y1, n );
                        Line10( x0, y0, x1, y0, n );
                        Line10( x1, y1, x0, y1, n );
                        Line10( x1, y1, x1, y0, n );
}
```

Listato 113. *Esecuzione di XOR su un rettangolo all'interno del buffer del video.*

Effetto elastico

Una tecnica correlata basata sull'operazione XOR è l'effetto elastico, nel quale un oggetto mobile resta ancorato ad un oggetto fisso tramite una linea retta. La tecnica viene chiamata "effetto elastico" in quanto la linea che collega i due oggetti sembra che si tenda mentre avviene lo spostamento. Il Listato 114 è analogo al Listato 113, ma produce lo spostamento di una linea elastica intorno al punto a (150,100).

```
/* Listato 114 */

#define                 Xmax  640             /* dimensioni schermo nella */
                                              /*  nella modalità 640x350 */
#define                 Ymax  350

main()
{
                        int  x0     = 150;    /* estremità fissa */
                                              /*  a 150,100 */
                        int  y0     = 100;
                        int  x      = 0;
                                              /* estremità mobile a 0,0 */
                        int  y      = 0;
                        int  n      = 12;     /* valore di pixel */

                        for( ; x<Xmax; x++ )  /* spostamento a destra */
                          XORLine( x0, y0, x, y, n )
                        for( --x; y<Ymax; y++ )
                                              /* spostamento verso il basso */
                          XORLine( x0, y0, x, y, n );

                        for( --y; x>=0; --x )
                                              /* spostamento a sinistra */
                          XORLine( x0, y0, x, y, n );

                        for( x++; y>=0; --y )
                                              /* spostamento verso l'alto */
                          XORLine( x0, y0, x, y, n );
}


XORLine                 ( x0, y0, x1, y1, n )
int                     x0,y0,x1,y1;                /* estremità */
int                     n;                          /* valore di pixel */
```

```
Line10( x0, y0, x1, y1, n );
                /* la linea è sullo schermo */
Line10( x0, y0, x1, y1, n );
                /* la linea è cancellata */
```

Listato 114. *Esecuzione di XOR su una linea all'interno del buffer del video.*

Spostamenti di blocchi di bit

Potete utilizzare XOR con uno spostamento di blocchi di bit per animare un qualsiasi gruppo di pixel. In ogni caso impiegate questa tecnica solo con un blocco di bit relativamente piccolo, dal momento che generalmente un blocco di bit contiene molti più pixel da tracciare e ritracciare di quanti ne contenga una linea o un rettangolo. Più tempo è necessario per manovrare il blocco di bit sullo schermo, più lenta sarà l'esecuzione della routine di visualizzazione.

Problemi con l'animazione tramite XOR

Gli oggetti animati con operazioni XOR presentano sempre uno sfarfallio. La ragione è ovvia: un oggetto è visibile solo dopo averlo sottoposto a XOR nel buffer. Il secondo XOR lo fa scomparire. Lo sfarfallio risultante attira l'attenzione sull'oggetto animato, cosa che potrebbe essere auspicabile, in particolare se l'oggetto viene ripetutamente sottoposto a XOR anche quando non lo state spostando. D'altro canto, lo sfarfallio potrebbe essere fonte di distrazione, in particolare sui monitor a colori dove l'oggetto sottoposto a XOR assume in successione due colori sgargianti.

E' talvolta possibile ridurre lo sfarfallio durante l'animazione con XOR introducendo una "pausa" software tra la prima e la seconda operazione XOR. Questa pausa può essere un loop vuoto, una chiamata ad una breve subroutine, o anche un'attesa del prossimo intervallo di soppressione verticale. In ogni caso, dal momento che l'oggetto sottoposto a XOR resta sullo schermo leggermente più a lungo, potrebbe presentare uno sfarfallio minore.

L'immagine animata può scomparire se il loop che effettua le operazioni XOR diventa inavvertitamente sincronizzato con il ciclo di ripristino dello schermo. In questa situazione, l'oggetto animato non è più visibile se entrambe le operazioni XOR avvengono al di fuori del relativamente breve intervallo di tempo durante il quale il quadro sta visualizzando la parte relativa del buffer del video. La soluzione di questo tipo di problema è particolarmente complessa in quanto coinvolge sia la velocità del vostro programma sia la dimensione dell'immagine animata.

Sovrapposizione di spostamenti di blocchi di bit

In alcune applicazioni, potete evitare i problemi di animazione con XOR ritracciando rapidamente un blocco di pixel su locazioni sovrapposte nel buffer del video (vedere Figura 120 ed il Listato 115). Il blocco di bit della Figura 120 ha un margine di pixel di

sfondo lungo il proprio bordo sinistro. Ogni volta che memorizzate il blocco di bit nel buffer del video, questo margine si sovrappone ai pixel di primo piano del blocco precedentemente tracciato. Senza questo margine, delle strisce indesiderate di pixel di primo piano seguirebbero il blocco di bit mentre viene spostato sullo schermo.

Sebbene siano abbastanza veloci per la maggior parte degli scopi, le routine di spostamento blocchi di bit trattate in questo capitolo sono troppo lente per applicazioni che richiedono alte prestazioni come ad esempio i video game "arcade" (quelli che si trovano nelle sale giochi). Potete adattare il codice in diversi modi per aumentare la velocità di animazione se siete disposti a sacrificare il loro approccio generico.

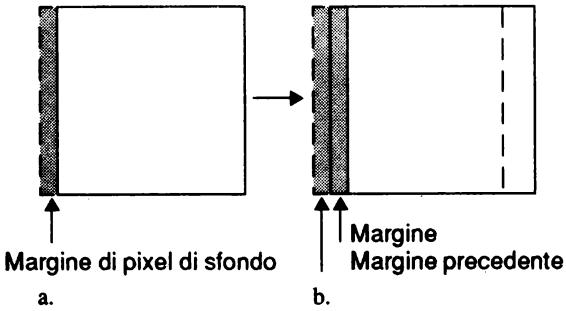


Figura 120. *Sovrapposizione di spostamenti di blocchi di bit. Il blocco di bit viene tracciato (Figura 120a), quindi ritracciato leggermente spostato a destra (Figura 120b). Il margine di pixel di sfondo ripristina lo sfondo mentre il blocco di bit viene "spostato" a destra.*

```
char far buf[(32/4)*21+5];        /* buffer del blocco di bit abbastanza
                                     grande per contenere un blocco di */
                                  /* 32 per 21 pixel */

Line(  1, 10, 21, 10, Fgd );      /* una freccia che punta a destra */
Line( 21,  0, 31, 10, Fgd );      /* in un blocco di 32 per 21 pixel */
Line( 21, 20, 31, 10, Fgd );
Line( 21,  0, 21, 20, Fgd );

GetBitBlock( 0, 0, 31, 20, buf );     /* copia il blocco di bit su */
                                      /* RAM di sistema */
for ( i = 0; i < 300; i++ )
  StoreBitBlock( buf, i, 0);          /* fa scorrere verso destra */
```

Listato 115. *Un programma per spostare un blocco di pixel utilizzando la tecnica della sovrapposizione.*

Una tecnica è quella di limitare le routine di blocco di bit ai blocchi di pixel allineati al byte (o, su CGA e HGC, allineati alla word). Ciò elimina molta della logica di maschera di bit e permette di utilizzare pienamente l'istruzione *MOVs* 80x86. Un altro ap-

proccio è quello di scrivere delle routine che gestiscano blocchi di bit di una dimensione fissa e predeterminata. Questo vi permette di sostituire alcuni loop iterativi nelle routine con sequenze ripetitive di codice in linea. Sfortunatamente, anche le routine di animazione altamente ottimizzate per CGA ed EGA raramente si avvicinano alla velocità che ci si potrebbe aspettare da un dispositivo hardware di visualizzazione di tipo "arcade".

Un cursore in modo grafico

Nelle modalità alfanumeriche, il cursore indica la locazione sullo schermo nella quale il programma attende il successivo input dell'utente. La maggior parte dei programmi in modalità alfanumerica si basa sul cursore lampeggiante generato dall'hardware per indicare la locazione di input corrente. Nelle modalità grafiche, invece, l'hardware non supporta un cursore: è il software che deve generarne uno.

L'implementazione di un cursore in una modalità grafica è un'operazione decisamente complessa, in quanto dovete tracciare direttamente nel buffer del video la forma che rappresenta il cursore e nel contempo mantenere i pixel che l'operazione sovrascrive. Potete realizzare tutto ciò in due modi: utilizzando XOR per visualizzare il cursore, o salvando e ripristinando il blocco di bit a cui il cursore si sovrappone.

XOR

Il metodo più semplice per visualizzare un cursore grafico è quello di sottoporlo a XOR nel buffer del video e successivamente di risottoporlo a XOR al di fuori del buffer del video. Questa tecnica è identica a quella utilizzata per animare le immagini grafiche e presenta gli stessi pro e contro.

Probabilmente l'effetto collaterale peggiore dell'utilizzo di XOR su un cursore grafico nel buffer del video è che il colore visualizzato per il cursore sottoposto a XOR può sostituirsi con quello di sfondo. Il cursore può scomparire su uno sfondo costituito da un modello o su uno sfondo di un colore simile a quello del cursore sottoposto a XOR.

La programmazione della tavolozza dei colori può prevenire questo problema. Ad esempio, la tavolozza dell'EGA nella Figura 121 è impostata presumendo che tutti i pixel che compongono il cursore abbiano un valore 8 (1000B) e che tutti i pixel preesistenti nel buffer del video abbiano un valore da 0 a 7. Con questa disposizione, l'esecuzione di XOR sul cursore nel buffer del video fa sì che esso venga sempre visualizzato con il valore di colore 3FH (bianco evidenziato). Lo svantaggio ovvio è che questa tecnica dimezza il numero di colori visualizzabili.

Spostamento di blocchi di bit

Un altro approccio è quello di creare una copia del blocco di bit dei pixel che il cursore sostituisce. Potete quindi cancellare il cursore ripristinando i pixel nel buffer del video dalla copia. Questa tecnica è interessante in quanto permette l'uso di quasiasi mezzo per il tracciamento del cursore.

Registro di tavolozza	Valore di colore
00H	0
01H	1
02H	2
03H	3
04H	4
05h	5
06h	6
07h	7
08H	3FH
09H	3FH
0AH	3FH
0BH	3FH
0CH	3FH
0DH	3FH
0EH	3FH
0FH	3FH

Figura 121. *Valori di tavolozza EGA per un cursore grafico bianco evidenziato sottoposto a XOR*

Un buon metodo per tracciare il cursore, una volta che avete creato una copia dei pixel sottostanti nel buffer del video, è quello di copiare la forma del cursore nel buffer con uno spostamento di blocchi di bit. Ovviamente, questa tecnica funziona meglio con un cursore rettangolare. Per tracciare un cursore di una forma arbitraria, utilizzate un processo a due fasi (vedere Figura 122). Per prima cosa azzerate un gruppo di pixel nella forma del cursore all'interno del buffer del video con un'operazione AND di blocco di bit. Successivamente, tracciate il cursore con un'operazione OR o XOR di blocco di bit.

CONSIGLIO

Ogni volta che utilizzate un cursore in modo grafico, dovete assicurarvi che il cursore venga cancellato prima di aggiornare il buffer del video. In caso contrario, il vostro programma potrebbe inavvertitamente aggiornare la parte di buffer del video che contiene l'immagine del cursore. Il successivo spostamento di cursore riporterà il contenuto del buffer allo stato precedente al tracciamento del cursore, lasciando un "buco" dove si trovava il cursore (vedere Figura 123).

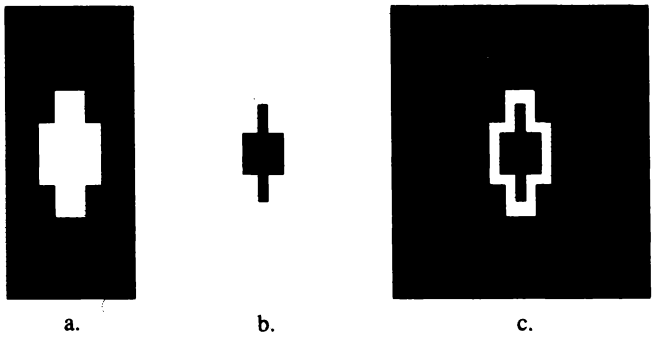


Figura 122. *Tracciamento di un cursore grafico con una tecnica a due fasi di mascheramento e sostituzione: per prima cosa, una maschera (Figura 122a) viene sottoposta a AND nel buffer del video. Successivamente la forma del cursore (Figura 122b) viene sottoposta a OR nel buffer per ottenere il risultato della Figura 122c.*

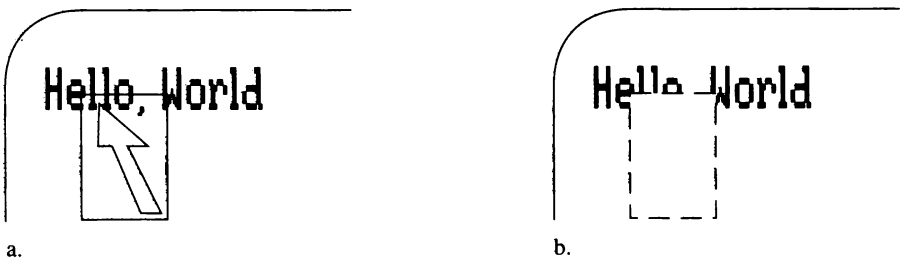


Figura 123. *Se un cursore grafico viene incidentalmente sovrascritto (Figura 123a), appare un "buco" quando il cursore viene cancellato (Figura 123b).*

12
Alcune tecniche avanzate di programmazione video

Un gestore di interrupt verticale
EGA e VGA - MCGA

Scorrimento dell'immagine su EGA e VGA
Posizionamento della finestra di schermo - Scorrimento
Ridimensionamento del buffer del video

Stratificazione dei piani di bit

Divisione schermo su EGA e VGA

L'interfaccia a penna ottica
Posizione della penna ottica - Commutatore della penna ottica
Determinazione delle modalità video Hercules

Il presente capitolo tratta alcune delle capacità meno sfruttate dei sistemi di visualizzazione dei PC e dei PS/2. La maggior parte dei programmatori non si occupano di queste funzioni hardware, in quanto vengono raramente utilizzate nella maggior parte dei programmi di visualizzazione. Tuttavia, ognuna di queste funzioni hardware si presta a tecniche di programmazione che possono essere utilizzate in applicazioni dove nient'altro è altrettanto efficace.

Niente di questo capitolo richiede una programmazione altamente specializzata o una qualche conoscenza magica dell'hardware. Ciononostante, quello che è richiesto è una conoscenza della programmazione in linguaggio Assembler 80x86 prima di addentrarvi nei dettagli del materiale esposto. La maggior parte del capitolo descrive tecniche di programmazione per EGA e VGA, ma quanto spiegato per l'interfaccia a penna ottica e per la stratificazione dei piani di bit riguarda anche gli adattatori Hercules.

Un gestore di interrupt verticale

Né l'interrupt né il gestore sono verticali, bensì questo termine indica che il CRTC dell'EGA, della VGA e dell'MCGA può generare un interrupt hardware all'inizio dell'intervallo di soppressione verticale, cioè all'inizio della linea di scansione successiva alla linea inferiore dei dati di buffer video visualizzati. Un gestore di interrupt per questo interrupt verticale può così aggiornare il buffer del video o programmare l'hardware di visualizzazione senza interferire con lo schermo.

L'interrupt viene generato dalla linea 2 di richiesta di interrupt (IRQ2). Il controller di interrupt programmabile (PIC) del computer viene impostato, durante l'avviamento a freddo del BIOS ROM, sulla mappa IRQ2 e sul vettore di interrupt 0AH, quindi un gestore di interrupt verticale dovrebbe essere progettato per gestire l'interrupt 0AH.

CONSIGLIO

Il controller di interrupt programmabile utilizzato su PC, PC/AT e PS/2 modelli 50, 60 e 80 dell'IBM è l'Intel 8259A; nel PS/2 modello 30 le stesse funzioni sono supportate all'interno di un chip esclusivo VLSI, il Support Gate Array di I/O. In tutti i casi, comunque, l'interfaccia di programmazione al PIC per la gestione degli interrupt verticali è identica.

EGA e VGA

Il numero di linea di scansione alla quale viene emesso l'interrupt è superiore di 1 rispetto al valore del registro di fine abilitazione visualizzazione verticale (12H) del CRTC. Il valore di questo registro specifica il numero di linee di scansione visualizzate

dei dati del buffer del video, in modo che il CRTC generi interrupt verticali all'inizio dell'intervallo di soppressione verticale.

I bit 4 e 5 del registro di ritracciamento verticale fine' (11H) del CRTC controllano se e quando il CRTC segnala un interrupt verticale. Impostate a 1 il bit 5 per abilitare il CRTC alla generazione dell'interrupt. Il bit 4 controlla un latch di 1 bit il cui stato appare nel bit 7 del registro di azzeramento stato (3C2H). Dovete azzerare il bit 4 per cancellare il latch di stato. Quando impostate a 1 il bit 4, il bit di stato del latch cambia da 0 a 1 non appena si verifica il successivo interrupt verticale e resta impostato a 1 fino a quando non azzerate nuovamente il latch.

Per utilizzare la funzione di interrupt verticale, dovete effettuare le seguenti azioni:

. Puntate il vettore di interrupt 0AH su un gestore di interrupt verticale.

. Abilitate IRQ2.

. Abilitate l'interrupt verticale.

La routine del Listato 116 mostra come procedere. Notate come questa routine sia coordinata con il gestore di interrupt stesso. La routine mantiene il vettore di interrupt 0AH in modo che il gestore di interrupt possa concatenarsi, se necessario, al precedente gestore, quindi la routine può alla fine ripristinare il precedente vettore di interrupt se il gestore di interrupt non è più necessario.

```
TITLE           'Listato 116'
                 NAME    VREGA
                 PAGE    55,132

;
; Nome:          VREGA
;
; Funzione:      Routine di servizio interrupt verticale per EGA e VGA
;
; Chiamante:     Microsoft C:
;
;                        int EnableISR0A();       /* ritorna 0 se
;                                                    installato bene */
;
;                        azzera DisableISR0A();
;

CRT_MODE         EQU     49h              ; indirizzi nell'area dati BIO
                                          ; del video
ADDR_6845        EQU     63h

DGROUP           GROUP   _DATA

_TEXT            SEGMENT byte public 'CODE'
                 ASSUME  cs:_TEXT,ds:DGROUP

ISR0A            PROC    far              ; gestore interrupt per INT 0Ah

                 push    ax               ; mantiene registri
                 push    dx
                 push    ds
```

```
                mov     ax,seg DGROUP
                mov     ds,ax           ; DS -> DGROUP
; determina se si è verificato un interrupt verticale

                mov     dx,3C2h         ; DX := porta di I/O per
                                        ;  azzeramento reg stato di input
                in      al,dx
                test    al,80h          ; test bit 7 del valore
                                        ;  del reg di stato
                jnz     L10             ; salta se si verifica
                                        ;  interrupt verticale

; non è un interrupt verticale quindi si collega al precedente gestore di
; interrupt

                pushf                   ; simula un'INT
                call    ds:PrevISR0A    ;  al precedente gestore
                                        ;  di INT 0Ah
                jmp     short Lexit

; gestisce un interrupt verticale

L10:            mov     dx,Port3x4      ; DX := 3B4h o 3D4h

                in      al,dx           ; AL := valore del reg
                            ;           ;  di indirizzo del CRTC
                push    ax              ; mantiene questo valore

                mov     ax,DefaultVREnd ; AH := valore di default per reg
                                        ;  ritrac. vert. fine
                                        ; AL := 11h (numero reg.)
                and     ah,11101111b    ; AH bit 4 := 0 (cancella latch
                                        ;  di interrupt)
                out     dx,ax           ; aggiorna registro ritrac. vert
                                        ;  fine
                jmp     $+2             ; attende che risponda il CRTC

; invia fine interrupt al controller di interrupt programmabile
; dell'Intel 8259A;per permettere l'emissione in successione di
; interrupt IRQ2

                mov     al,20h          ; porta di I/O 8259A
                out     20h,al          ; invia fine interrupt non
                                        ;  specifico a 8259A
                jmp     $+2             ; attende che risponda il PIC
                sti                     ; abilita interrupt

; esegue operazioni utili ...

                inc     word ptr_VRcount  ; incrementa un contatore

; abilita CRTC in modo che generi un altro interrupt

                cli                     ; disabilita interrupt
                mov     ax,DefaultVREnd ; AH := valore di default del reg
                                        ;  ritrac. vert. fine
```

```
                                        ; AL := 11h (numero reg.)
                and     ah,11011111b    ; AH bit 5 := 0 (abilita int
                                        ;  verticale)
                or      ah,00010000b    ; AH bit 4 := 1 (abilita latch di
                                        ;  int)
                out     dx,ax
                jmp     $+2

                pop     ax
                out     dx,al           ; ripristina precedente valore
                                        ;reg indirizzo

Lexit:          pop     ds              ;ripristina registri ed esce
                pop     dx
                pop     ax
                iret

ISR0A           ENDP

;
; EnableISR0A -- abilita gestore interrupt verticale
;
                PUBLIC  _EnableISR0A
_EnableISR0A    PROC    near

                push    bp              ; mantiene registri del chiamante
                mov     bp,sp
                push    si
                push    di

                mov     ax,40h
                mov     es,ax           ; ES -> area dati BIOS del video

; salva valori di default del registro del CRTC

                mov     dx,es:[ADDR_6845]  ; DX := porta indirizzo del
                                           ;  CRTC
                mov     Port3x4,dx      ; salva indirizzo di porta

                mov     ax,1A00h        ; AH := 1AH (numero funzione INT
                                        ;  10H)
                                        ; AL := 0 (legge combinazione di
                                        ;  visualizzazione)
                int     10h             ; AL := 1AH se funzione 1AH è
                                        ;  supportata
                                        ; BL := attiva sistema di
                                        ;  visualizzazione
                cmp     al,1Ah
                jne     L20             ; salta se non è VGA

                cmp     bl,7
                je      L21             ; salta se è VGA

                cmp     bl,8
                je      L21             ; salta se è VGA

                mov     ax,0FFFFh       ; ritorna 0FFFFh se non è né EGA
                                        ;  né VGA
```

```
                jmp     short L23

; rileva valore di default per registro ritracciamento verticale fine per
; EGA

L20:            mov     al,es:[CRT_MODE]  ; AL := numero modalità BIOS ;
                                          ;  video
                mov     bx,offset DGROUP:EGADefaultVals
                xlat                      ; AL := valore di default per
                                          ;  reg ritrac. vert. fine
                jmp     short L22

; rileva valore di default per registro ritracciamento verticale fine per
; VGA

L21:            mov     al,VREndReg     ; AL := numero reg. ritrac. vert.
                                        ;  fine
                out     dx,al
                inc     dx              ; DX := 3B5H o 3D5H
                in      al,dx           ; AL := valore corrente per il
                                        ;  registro

L22:            mov     VREndValue,al   ; salva questo valore

; salva vecchio vettore di interrupt 0Ah

                mov     ax,350Ah        ; AH := 35H (numero funzione INT
                                        ;  21h)
                                        ; AL := 0AH (numero interrupt)
                int     21h             ; ES:BX := precedente vettore INT
                                        ;  0AH

                mov     word ptr PrevISR0A,bx
                mov     word ptr PrevISR0A+2,es ; salva vettore precedente

; aggiorna vettore di interrupt 0AH con indirizzo di questo gestore

                push    ds              ; mantiene DS
                mov     dx,offset ISR0A
                push    cs
                pop     ds              ; DS:DX - >ISR0A
                mov     ax,250Ah        ; AH := 25H (numero funzione INT
                                        ;  21H)
                                        ; AL := 0AH (numero interrupt)
                int     21h             ; aggiorna vettore INT 0AH
                pop     ds              ; ripristina DS

; abilita IRQ2 azzerando il bit 2 del registro di maschera dell'8259A

                 cli                    ; azzera interrupt
                mov     dx,21h          ; DX := registro di maschera 8259A
                in      al,dx           ; AL := valore registro di
                             ;            maschera
                and     al,11111011b    ; azzera bit 2
                out     dx,al
```

```
; abilita interrupt verticali

                mov     dx,Port3x4      ; DX := 3B4H o 3D4H
                mov     ax,DefaultVREnd

and     ah,11001111b
                out     dx,ax           ; azzera bit 4 e 5 del reg di
                                ;           ritrac. vert. fine
                jmp     $+2             ; attende risposta dal CRTC
                or      ah,00010000b
                out     dx,ax           ; imposta bit 4
                jmp     $+2
                sti                     ; abilita interrupt

                xor     ax,ax           ; AX := 0 (ritorna valore)

L23:            pop     di              ; ripristina registri ed esce
                pop     si
                mov     sp,bp
                pop     bp
                ret

_EnableISR0A    ENDP

;
; DisableISR0A -- disabilita gestore interrupt verticale
;
                PUBLIC  _DisableISR0A
_DisableISR0A   PROC    near

                push    bp
                mov     bp,sp
                push    si
                push    di
                push    ds

; disabilita interrupt verticali

                cli                     ; disabilita interrupt
                mov     dx,Port3x4
                mov     ax,DefaultVREnd
                out     dx,ax           ; ripristina reg ritrac. vert
                                        ;  fine
                jmp     $+2
                sti                     ; abilita interrupt

; ripristina precedente gestore interrupt 0Ah

                lds     dx,PrevISR0A    ; DS:DX := precedente vettore INT
                                        ;  0AH

                mov     ax,250Ah        ; AH := 25H (numero funzione INT
                                        ;  21H)
                                        ; AL := 0AH (numero interrupt)
                int     21h

                pop     ds              ; ripristina registri ed esce
                pop     di
```

```
                pop     si
                mov     sp,bp
                pop     bp
                ret

_DisableISR0A   ENDP
_TEXT           ENDS

_DATA           SEGMENT word public 'DATA'

                EXTRN   _VRcount:word   ; dichiarata nel chiamante C

PrevISR0A       DD      ?               ; salva area per vecchio
                                        ;   vettore int 0Ah
Port3x4         DW      ?               ; 3B4h o 3D4h

DefaultVREnd    LABEL   word
VREndReg        DB      11h             ; numero reg ritrac. vert. fine
VREndValue      DB      ?               ; valore di default per reg
                                        ;  ritrac. vert. fine

EGADefaultVals  DB      2Bh,2Bh,2Bh,2Bh,24h,24h,23h,2Eh ; valori di
               ;                                          default per
                DB      00h,00h,00h,00h,00h,24h,23h,2Eh ;  reg ritrac.
                                                        ;   vert. fine EGA

                DB      2Bh

_DATA           ENDS

                END
```

Listato 116. *Gestione degli interrupt verticali su EGA e VGA.*

Il gestore stesso, nella procedura *ISR0A*, ottiene il controllo ogni volta che si verifica l'interrupt 0AH. Per distinguere tra l'interrupt verticale hardware su IRQ2 ed un eventuale interrupt software 0AH, il gestore esamina il bit 7 del registro di azzeramento del registro di stato di input. Se questo bit è 1, significa che si è verificato un interrupt verticale e il gestore continua il suo lavoro. Se il bit è 0, significa che non si è verificato un interrupt verticale, quindi il gestore si collega al precedente gestore di interrupt 0AH.

CONSIGLIO

Un inconveniente derivante dall'uso dell'interrupt verticale è che qualsiasi interrupt hardware su IRQ2 provoca l'impostazione del bit di stato del registro di azzeramento registro di stato di input. Di conseguenza, sebbene il bit di stato possa essere usato per rilevare l'interrupt software 0AH, un gestore di interrupt non può distinguere tra interrupt verticali EGA ed interrupt IRQ2 generati da altri dispositivi hardware a meno che gli altri dispositivi hardware non possano essere verificati in modo affidabile. Dal momento che alcuni altri

adattatori per PC IBM possono utilizzare IRQ2 (ad esempio, la versione bus del mouse Microsoft), potete impiegare in modo affidabile l'interrupt verticale solo quando siete certi dell'esatta configurazione hardware del PC sul quale sta operando il vostro programma.

Dopo che il gestore ha rilevato un interrupt verticale (cioè, il bit 7 del registro di azzeramento registro di stato di input è impostato a 1), emette un'istruzione non specifica di fine interrupt (EOI) al controller di interrupt in modo che i successivi interrupt IRQ2 possano essere elaborati. Il rientro non è un problema, in quanto interrupt verticali addizionali non verranno segnalati fino a che il gestore stesso non azzera e riabilita il latch di stato. Una volta che l'EOI è stata emessa, il gestore è libero di eseguire alcune azioni utili. In questo esempio, esso incrementa semplicemente un contatore, e subito prima di uscire, riprogramma il registro di ritracciamento verticale fine per abilitare il successivo interrupt verticale.

L'esempio del Listato 117 mostra come potreste integrare un gestore di interrupt verticale in un programma ad alto livello. L'esempio è intenzionalmente semplice. Esso non effettua nulla se non il conteggio di un numero determinato di interrupt verticali e la visualizzazione di un messaggio. Naturalmente, il vostro gestore di interrupt verticale potrebbe effettuare azioni più complicate del semplice aggiornamento di una variabile. Ad esempio, potreste effettuare dell'animazione aggiornando il buffer del video ogni volta che si verifica un interrupt. Potreste anche aggiornare il controller di attributo e del CRT per produrre un effetto di scorrimento utilizzando le tecniche descritte più avanti nel corso del presente capitolo.

```
/* Listato 117 */

int   VRcount = 0;            /* contatore di interrupt verticale */

main()
{
      if ( EnableISR0A() )
      {
        printf( "\nNon è possibile abilitare gestore di interrupt
                  verticale\n" );
        exit( 1 );
      }


      while (VRcount < 600)
        printf( "\015Numero di interrupt verticali:  %d", VRcount );

      DisableISR0A();
}
```

Listato 117. *Uso di un gestore di interrupt verticale in un programma in C.*

CONSIGLIO

Il supporto hardware per la funzione di interrupt verticale può variare. L'adattatore VGA dell'IBM, ad esempio, non supporta assolutamente gli interrupt verticali. Su alcuni cloni EGA, la polarità del bit 7 del registro di azzeramento registro di stato di input è l'opposto di quella del bit EGA equivalente; ciò significa che un interrupt verticale si verifica quando il bit 7 è 0 (i produttori per conto terzi degli adattatori compatibili EGA non emulano sempre ogni dettaglio del progetto hardware a volte imperscrutabile dell'EGA). Per assicurarvi che il vostro gestore di interrupt funzioni correttamente sui cloni EGA, determinate la polarità del bit di stato quando il bit si trova in uno stato conosciuto e sistemate di conseguenza il vostro test per l'interrupt verticale.

MCGA

Un gestore di interrupt verticale per l'MCGA, come quello del Listato 118, è analogo al gestore per EGA e VGA. Su MCGA, il registro di controllo dell'interrupt (11H) contiene i bit di controllo e di stato usati per impostare e rilevare un interrupt verticale. L'azzeramento del bit 5 del registro di controllo interrupt abilita l'MCGA alla generazione di un interrupt verticale. L'azzeramento del bit 4 cancella il latch di stato dell'interrupt. L'impostazione a 1 del bit 4 permette all'MCGA di rilevare successivi interrupt. Il bit 6 è il bit di stato dell'interrupt. L'MCGA imposta a 1 questo bit per indicare che si è verificato un interrupt verticale.

```
TITLE           'Listato 118'
                NAME    VRMCGA
                PAGE    55,132

;
; Nome:         VRMCGA
;
; Funzione:     Routine di servizio interrupt verticale per MCGA
;
; Chiamante:    Microsoft C:
;
;                       int EnableISR0A();      /* ritorna 0 se */
;                                               /*  installato bene */
;
;                       azzera DisableISR0A();
;

ADDR_6845       EQU     63h

DGROUP          GROUP   _DATA

_TEXT           SEGMENT byte public 'CODE'
```

```
                ASSUME  cs:_TEXT,ds:DGROUP

ISR0A           PROC    far             ; gestore interrupt per INT 0Ah

                push    ax              ; mantiene registri
                push    dx
                push    ds

                mov     ax,seg DGROUP
                mov     ds,ax           ; DS -> DGROUP

                mov     dx,Port3x4      ; DX := numero reg indirizzo CRTC
                in      al,dx
                push    ax              ; mantiene valore reg
                                         ; indirizzo CRTC
; determina se si è verificato un interrupt verticale

                mov     al,IConReg      ; AL := numero registro
                out     dx,al
                jmp     $+2             ; attende risposta da MCGA
                inc     dx              ; DX := 3D5H
                in      al,dx           ; AL := valore corrente registro
                                        ;  controllo interrupt
                dec     dx
                test    al,40h          ; test bit 6
                jnz     L10             ; salta se si verifica interrupt
                                        ;  verticale

; non è un interrupt verticale quindi si collega al precedente gestore di
interrupt

                pushf                   ; simula un'INT al
                call    ds:PrevISR0A    ;  precedente gestore di INT 0Ah
                jmp     short Lexit

; gestisce un interrupt verticale

L10:            mov     ax,DefaultICont ; AH := valore di default per reg
                                        ;  controllo interrupt
                                        ; AL := 11h (numero reg.)
                and     ah,11101111b    ; AH bit 4 := 0 (cancella latch
                                        ;  di interrupt)
                out     dx,ax           ; aggiorna reg. controllo
                                        ;  interrupt
                jmp     $+2             ; attende che risponda l'MCGA

; invia fine interrupt al controller di interrupt programmabile
;  per permettere l'emissione in successione di interrupt IRQ2

                mov     al,20h          ; porta di I/O del PIC
                out     20h,al          ; invia fine interrupt non
                                         ; specifico al PIC
                jmp     $+2             ; attende che risponda il PIC
                sti                     ; abilita interrupt

; esegue operazioni utili ...
```

```
                inc     word ptr _VRcount ; incrementa un contatore

; abilita CRTC in modo che generi un altro interrupt

                cli                       ; disabilita interrupt
                mov     ax,DefaultICont   ; AH := valore di default del reg
                                          ; controllo interrupt
                                          ; AL := 11h (numero reg.)
                and     ah,11011111b      ; AH bit 5 := 0 (abilita int
                             verticale)
                or      ah,00010000b      ; AH bit 4 := 1 (abilita latch
                                          ;  di int)
                out     dx,ax
                jmp     $+2

Lexit:          pop     ax
                out     dx,al             ; ripristina precedente valore
                                          ;  3D4H

                pop     ds                ; ripristina registri ed esce
                pop     dx
                pop     ax
                iret

ISR0A           ENDP

;
; EnableISR0A -- abilita gestore interrupt verticale
;
                PUBLIC  _EnableISR0A
_EnableISR0A    PROC    near

                push    bp                ; mantiene registri del chiamante
                mov     bp,sp
                push    si
                push    di

                mov     ax,40h
                mov     es,ax             ; ES - >area dati BIOS del video

; salva valori di default del registro del CRTC

                mov     dx,es:[ADDR_6845]       ; DX := porta indirizzo del
                                                ;  CRTC
                mov     Port3x4,dx              ; salva indirizzo di porta

                mov     ax,1A00h                ; AH := 1AH (numero funzione
                                                ;  INT 10H)
                                                ; AL := 0 (legge combinaz
                                                ;  di visualizzazione)
                int     10h                     ; AL := 1AH se funzione 1AH
                                                ;  è supportata
                                                ; BL := sottosistema di
                                                ;  visualizzazione attivo
                cmp     al,1Ah
                jne     L20             ; salta se non è MCGA
```

```
                cmp     bl,0Bh
                je      L21                 ; salta se è MCGA

                cmp     bl,0Ch
                je      L21                 ; salta se è MCGA

L20:            mov     ax,0FFFFh           ; ritorna 0FFFFh se non è MCGA
                jmp     short L23

; rileva valore di default per registro controllo interrupt MCGA

L21:            mov     al,IContReg         ; AL := numero reg. controllo
                                            ;  interrupt
                cli
                out     dx,al
                jmp     $+2
                inc     dx                  ; DX := 3D5H
                in      al,dx               ; AL := valore corrente per il
                                            ;  registro
                sti

                mov     IContValue,al       ; salva questo valore

; salva vecchio vettore di interrupt 0Ah

                mov     ax,350Ah            ; AH := 35H (numero funzione
                                            ;  INT 21h)
                                            ; AL := 0AH (numero interrupt)
                int     21h                 ; ES:BX := precedente vettore
                                            ;  INT 0AH

                mov     word ptr PrevISR0A,bx
                mov     word ptr PrevISR0A+2,es ; salva vettore precedente

; aggiorna vettore di interrupt 0AH con indirizzo di questo gestore

                push    ds                  ; mantiene DS
                mov     dx,offset ISR0A
                push    cs
                pop     ds                  ; DS:DX -> ISR0A
                mov     ax,250Ah            ; AH := 25H (numero funzione
                                            ;  INT 21H)
                                            ; AL := 0AH (numero interrupt)
                int     21h                 ; aggiorna vettore INT 0AH
                pop     ds                  ; ripristina DS

; abilita IRQ2 azzerando il bit 2 del registro di maschera del PIC

                cli                         ; azzera interrupt
                mov     dx,21h              ; DX := registro di
                                            ;  maschera del PIC
                in      al,dx               ; AL := valore registro
                                            ;  di maschera
                and     al,11111011b        ; azzera bit 2
                out     dx,al

; abilita interrupt verticali
```

```
                mov     dx,Port3x4        ; DX := porta indirizzo del CRTC
                mov     ax,DefaultIcont

                and     ah,11001111b
                out     dx,ax             ; azzera bit 4 e 5 del reg
                                          ;controllo int
                jmp     $+2               ; attende risposta dall'MCGA
                or      ah,00010000b
                out     dx,ax             ; imposta bit 4
                jmp     $+2
                sti                       ; abilita interrupt

                xor     ax,ax             ; AX := 0 (ritorna valore)

L23:            pop     di                ; ripristina registri ed esce
                pop     si
                mov     sp,bp
                pop     bp
                ret

_EnableISR0A    ENDP

;
; DisableISR0A -- disabilita gestore interrupt verticale
;
                PUBLIC  _DisableISR0A
_DisableISR0A   PROC    near


                push    bp
                mov     bp,sp
                push    si
                push    di
                push    ds

; disabilita interrupt verticali

                cli                       ; disabilita interrupt
                mov     dx,Port3x4
                mov     ax,DefaultICont
                out     dx,ax             ; ripristina reg controllo
                                          ;  interrupt
                jmp     $+2
                sti                       ; abilita interrupt

; ripristina precedente gestore interrupt 0Ah

                lds     dx,PrevISR0A      ; DS:DX := precedente vettore
                                          ;  INT 0AH
                mov     ax,250Ah          ; AH := 25H (numero funzione
                                          ;  INT 21H)
                                          ; AL := 0AH (numero interrupt)
                int     21h

                pop     ds                ; ripristina registri ed esce
                pop     di
                pop     si
```

```
                mov     sp,bp
                pop     bp
                ret

_DisableISR0A   ENDP

_TEXT           ENDS

_DATA           SEGMENT word public 'DATA'
                EXTRN   _VRcount:word     ; dichiarata nel chiamante C

PrevISR0A       DD      ?                 ; salva area per vecchio vettore
                                          ;  int 0Ah
Port3x4         DW      ?                 ; 3B4h o 3D4h

DefaultICont    LABEL   word
IContReg        DB      11h               ; numero reg controllo interrupt
IContValue      DB      ?                 ; valore di default per
                                          ;  reg controllo interrupt

_DATA           ENDS
                END
```

Listato 118. *Gestione degli interrupt verticali su MCGA.*

CONSIGLIO

Su EGA e MCGA, se un gestore di interrupt verticale ottiene il controllo mentre è in esecuzione una funzione (INT 10H) del BIOS del video, il gestore di interrupt potrebbe inavvertitamente compromettere la programmazione del CRTC del BIOS. La ragione può essere trovata in una subroutine del BIOS IBM di questi sistemi di visualizzazione. Questa subroutine viene richiamata da diverse routine del BIOS del video per eseguire l'output di porta di I/O verso i registri hardware del video, tra cui i registri del controller del CRT, del sequencer, del controller grafico e del controller di attributo.

Sfortunatamente, questa subroutine è accessibile agli interrupt. Essa contiene una sequenza di due scritture di porta a 8 bit (*OUT DX,AL*). La prima *OUT* carica il registro di indirizzo indicato. La seconda *OUT* scrive un byte di dati sul corrispondente registro dati. Se si verifica un interrupt tra le due scritture di porta, e se il gestore di interrupt stesso scrive sulla stessa porta, la seconda scrittura di porta della subroutine del BIOS potrebbe non essere convalidata.

Per evitare questa situazione su EGA e MCGA, i gestori di interrupt verticale dei Listati 116 e 118 leggono il valore del registro di indirizzo del CRTC alla porta 3D4H (3B4H su EGA con un monitor monocromatico). Su EGA questo valore è

leggibile solo per circa 15 millisecondi dopo che la porta è stata scritta, ma questo intervallo di tempo è sufficiente per il gestore di interrupt verticale per effettuare la lettura e mantenere il valore del registro di indirizzo del CRTC. Il gestore può quindi ripristinare il valore prima di ritornare dall'interrupt.

Scorrimento dell'immagine su EGA e VGA

Il buffer del video da 256 KB del'EGA e della VGA può memorizzare diversi schermi di dati. Di conseguenza, in un certo senso, ciò che viene visualizzato rappresenta una "finestra dello schermo", una sorta di finestra hardware che si affaccia sul contenuto del buffer del video.

Posizionamento della finestra di schermo

Su un adattatore come l'MDA o il CGA, i registri di inizio indirizzo del controller del CRT controllano quale area del buffer del video viene visualizzata. Dal momento che questi registri contengono un offset di byte nel buffer del video, potete controllare la posizione della finestra di schermo solo facendo riferimento al byte più vicino. D'altro canto, il controller del CRT su EGA e VGA può posizionare l'inizio della finestra di schermo ad una qualsiasi posizione di pixel indicata.

Nelle modalità grafiche il contenuto dei registri di inizio indirizzo alto e di inizio indirizzo basso (0CH e 0DH) del CRTC posizionano la finestra di schermo sull'offset di byte più vicino nel buffer video. Il contenuto del registro di scansione riga preimpostata (08H) del CRTC e il registro di scorrimento pixel orizzontale (13H) del controller di attributo "mettono a punto" la posizione della finestra di schermo pixel per pixel (vedere Figura 124).

Quando modificate la posizione della finestra di schermo pixel per pixel l'immagine visualizzata sembra scorrere sullo schermo. Un metodo comodo per fare ciò è quello di scrivere una routine che ponga la finestra di schermo ad una posizione di pixel specificata e quindi che richiami la routine iterativamente dall'interno di un loop. Questa routine, come mostrato nel Listato 119, deve distinguere tra modalità alfanumeriche e modalità grafiche. Essa deve anche gestire una matrice di carattere larga 9 pixel nelle modalità alfanumeriche monocromatiche VGA ed EGA.

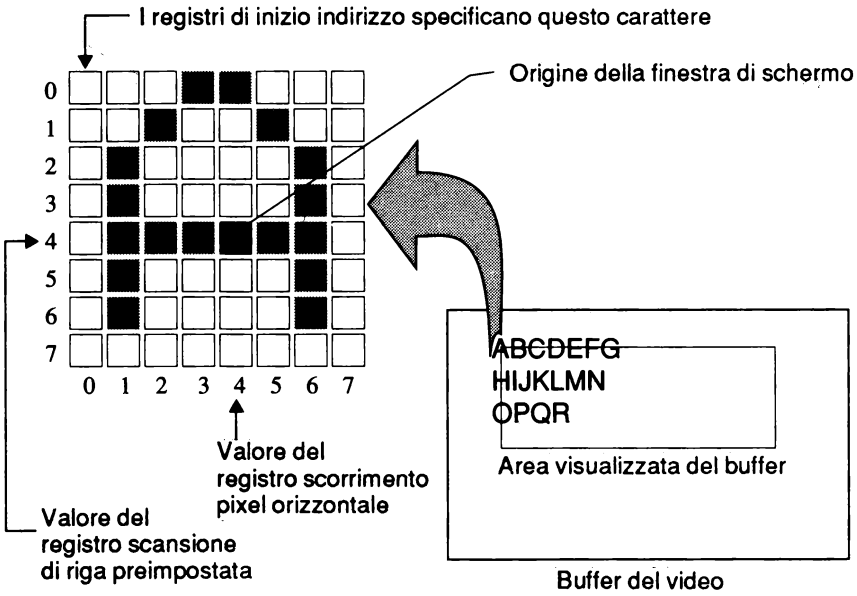


Figura 124. *Controllo dell'area visualizzata del buffer del video nelle modalità alfanumeriche.*

```
            TITLE 'Listing 12-4'
            NAME  ScreenOrigin
            PAGE  55,132
;
; Nome:     ScreenOrigin
;
; Funzione: Imposta origine di schermo su EGA e VGA
;
; Chiamante:Microsoft C:
;
;                 azzera ScreenOrigin(x,y);
;
;                 int   x,y;  /* coordinate di pixel x,y */
;

ARGx  EQU   [bp+4]
ARGy  EQU   [bp+6]


CRT_MODE    EQU   49h         ; indirizzi nell'area dati del BIOS del
                              ;  video
ADDR_6845   EQU   63h
POINTS      EQU   85h
BIOS_FLAGS  EQU   89h

DGROUP      GROUP _DATA

_TEXT       SEGMENT     byte public 'CODE'
            ASSUME      cs:_TEXT,ds:DGROUP
```

```
            PUBLIC      _ScreenOrigin
_ScreenOrigin     PROC  near

            push  bp          ; mantiene registri del chiamante
            mov   bp,sp
            push  si
            push  di

            mov   ax,40h
            mov   es,ax       ; ES -> area dati del BIOS del video
            mov   cl,es:[CRT_MODE]

            mov   ax,ARGx     ; AX := coordinata x del pixel
            mov   bx,ARGy     ; BX := coordinata y del pixel
            cmp   cl,7
            ja    L01         ; salta se è in modalità grafica
            je    L02         ; salta se è in modal. alfan. monocrom.

            test  byte ptr es:[BIOS_FLAGS],1
            jnz   L02         ; salta se è VGA
            jmp   short L03

; impostazione per modalità grafiche (8 pixel per byte)

L01:        mov   cx,8        ; CL := 8 (pixel visualizzati per byte)
                              ; CH := 0 (valore per scansione riga
                              ;  preimpostata)
            div   cl          ; AH := offset di bit nel byte
                              ; AL := offset di byte nella riga di
                              ;  pixel
            mov   cl,ah       ; CL := offset di bit (per scorrimento
                              ;  pixel oriz.)
            xor   ah,ah
            xchg  ax,bx       ; AX := y
                              ; BX := offset di byte nella riga di
                              ;  pixel

            mul   word ptr _BytesPerRow
                              ; AX := offset di byte dell'inizio riga
            jmp   short L05

; impostazione per modalità alfanumeriche VGA e per modalità alfanumerica
;  monocromatica EGA (9 pixel per byte)

L02:                          ; routine per modalità alfanumeriche
            mov   cx,9        ; CL := 9 (pixel visualizzati per byte)
                              ; CH := 0
            div   cl          ; AH := offset di bit nel byte
                              ; AL := offset di byte nella riga di
                              ;  pixel
            dec   ah          ; AH := -1, 0-7
            jns   L04         ; salta se offset di bit è da 0 a 7
            mov   ah,8        ; AH := 8
            jmp   short L04

; impostazione per modalità alfanumeriche a colori EGA (8 pixel per byte)
```

```
L03:        mov   cx,8          ; CL := 8 (pixel visualizzati per byte)
                                ; CH := 0
            div   cl            ; AH := offset di bit nel byte
                                ; AL := offset di byte nella riga di
                                ;  pixel

L04:        mov   cl,ah         ; CL := valore per reg scorrimento
                                ;  pixel oriz.
            xor   ah,ah
            xchg  ax,bx         ; AX := y
                                ; BX := offset di byte nella riga
            div   byte ptr es:[POINTS]    ; AL := riga di caratteri
                                ; AH := linea di scansione nella
                                ;  matrice di car.
            x207,ch             ; AX := riga di carattere
                                ; CH := linea di scansione (valore per
                                ;      reg scansione riga preimpostata)
            mul   word ptr _BytesPerRow   ; AX := offset di byte della
                                ;  riga di caratteri
            shr   ax,1          ; AX := offset di word di riga di
                                ;  caratteri

L05:        call  SetOrigin

            pop   di            ; ripristina registri ed esce
            pop   si
            mov   sp,bp
            pop   bp
            ret

_ScreenOrigin     ENDP


SetOrigin   PROC  near          ; Chiamante:  AX = offset della riga di
                                ;                   caratteri
                                ;             BX = offset di byte nella
                                ;                   riga
                                ;             CH = valore di scansione
                                ;                   riga preimpostata
                                ;             CL = valore di scorrim.
                                ;                    pixel orizzont.

            add   bx,ax         ; BX := offset di buffer

            mov   dx,es:[ADDR_6845] ; porta di I/O del CRTC (3B4H o 3D4H)
                  add   dl,6        ; porta di stato video (3BAH o 3DAH)

; aggiorna registri inizio indirizzo alto e basso

L20:        in    al,dx         ; attende inizio ritracc. verticale
            test  al,8
            jz    L20

L21:        in    al,dx         ; attende fine ritracciamento verticale
            test  al,8
            jnz   L21
            cli                 ; disabilita interrupt
            sub   dl,6          ; DX := 3B4H o 3D4H
```

```
            mov   ah,bh        ; AH := valore di inizio indirizzo alto
            mov   al,0Ch       ; AL := numero reg inizio indiriz. alto
            out   dx,ax        ; aggiorna questo registro
            mov   ah,bl        ; AH := valore per inizio indir. basso
            inc   al           ; AL := numero reg. inizio indir. basso
            out   dx,ax        ; aggiorna questo registro
            sti                ; abilita interrupt
            add   dl,6         ; DX := porta di stato del video

L22:        in    al,dx        ; attende inizio ritracc. verticale
            test  al,8
            jz    L22
            cli                ; disabilita interrupt

            sub   dl,6         ; DX := 3B4H o 3D4H
            mov   ah,ch        ; AH := valore del reg. scansione riga
                               ;  preimpostata
            mov   al,8         ; AL := numero reg scansione riga
                               ;  preimpostata
            out   dx,ax        ; aggiorna questo registro
            mov   dl,0C0h      ; DX := 3C0h (porta del controller di
                               ;  attributo)
            mov   al,13h OR 20h     ; AL bit 0-4 := numero registro
                                    ;  scorrimento pixel oriz.
                               ; AL bit 5 := 1
            out   dx,al        ; scrive reg indirizzo controller di
                               ;  attributo
                               ;  (flip-flop indirizzo del controller
                               ;  di attributo è stato azzerato da
                               ;  IN a L22.)
            mov   al,cl        ; AL := valore del reg. scorrimento
                               ;  pixel oriz.
            out   dx,al        ; aggiorna questo registro
            sti                ; riabilita interrupt
            ret

SetOrigin   ENDP

_TEXT       ENDS

_DATA       SEGMENT     word public 'DATA'

            EXTRN _BytesPerRow:word ; byte per riga di pixel

_DATA       ENDS

            END
```

Listato 119. *Impostazione dell'origine dello schermo su EGA e VGA.*

ScreenOrigin() accetta come input le coordinate x e y del pixel che identifica l'origine (l'angolo superiore sinistro) dello schermo. La routine per prima cosa aggiorna i registri di inizio indirizzo del CRTC. In effetti, questo posiziona lo schermo sul pixel superiore sinistro del carattere che contiene l'origine nelle modalità alfanumeriche, o sul

pixel posto all'estrema sinistra nel byte che contiene l'origine nelle modalità grafiche. Successivamente *ScreenOrigin()* posiziona lo schermo virtuale esattamente aggiornando i registri di scorrimento pixel orizzontale e di scansione riga preimpostata.

Il contenuto del registro di scorrimento pixel orizzontale del controller di attributo corrisponde all'offset di bit del pixel posto nell'angolo superiore sinistro dello schermo. Il valore da memorizzare in questo registro è quindi

```
x MOD 8
```

Nel caso di caratteri a 9 pixel nelle modalità alfanumeriche VGA e nella modalità monocromatica 80 per 25 su EGA, il valore è

```
(x + 8) MOD 9
```

Il registro di scorrimento pixel orizzontale è programmato allo stesso modo sia nelle modalità alfanumeriche sia in quelle grafiche. Questo non è il caso però del registro di scansione riga preimpostata, che controlla la posizione verticale dell'inizio dello schermo.

Nelle modalità alfanumeriche, il numero di righe di pixel visualizzate per ogni riga di caratteri nel buffer del video dipende dall'altezza della matrice di carattere visualizzato. Questo è il valore memorizzato come *POINTS* nell'area dati di visualizzazione del BIOS ROM. I registri di inizio indirizzo posizionano lo schermo virtuale su un carattere particolare nel buffer del video ed il registro di scansione riga preimpostata indica quale linea nella matrice di carattere contiene l'origine dello schermo virtuale. Il registro di scansione riga preimpostata contiene perciò un valore tra 0 (la linea superiore del carattere) e *POINTS-1* (la linea inferiore). Nelle modalità grafiche, i pixel di ogni byte nel buffer del video hanno una corrispondenza uno a uno con i pixel sullo schermo, quindi il registro di scansione riga preimpostata contiene sempre 0.

Per evitare interferenze con lo schermo, gli aggiornamenti dei registri di scorrimento pixel orizzontale, scansione riga preimpostata e inizio indirizzo dovrebbero essere sincronizzati con il ciclo di ripristino dello schermo. Il registro di scorrimento pixel orizzontale deve essere aggiornato durante l'intervallo di soppressione verticale. D'altro canto, il CRTC verifica i valori dei registri di inizio indirizzo e di scansione riga preimpostata all'inizio del ritracciamento verticale, quindi questi registri dovrebbero essere aggiornati quando il ritracciamento verticale non è attivo.

Scorrimento

La routine del Listato 120 mostra come potete richiamare *ScreenOrigin()* per far scorrere lo schermo verso l'alto e verso il basso o orizzontalmente nel buffer del video. Dal momento che la posizione dello schermo virtuale cambia sempre durante un intervallo di soppressione verticale, l'effetto di scorrimento è senza scatti, senza interferenze con lo schermo.

```
Pan( x0, y0, x1, y1 )
int            x0,y0;            /* coordinate di pixel iniziali */
int            x1,y1;            /* coordinate di pixel finali */
{
               int    i = x0;
               int    j = y0;
               int    Xinc,Yinc; /* incrementi orizzontal e verticali */

               if ( x0 < x1 )    /* calcola segni degli incrementi */
                 Xinc = 1;
               else
                 Xinc = -1;

               if ( y0 < y1 )
                 Yinc = 1;
               else
                 Yinc = -1;

               while ( (i != x1) // (j != y1) )
               {
                  if (i != x1)   /* calcola origine schermo successivo */
                    i += Xinc;
                  if (j != y1)
                    j += Yinc;

                  ScreenOrigin(i,j); /* sposta origine schermo */
               }
}
```

Listato 120. *Una routine per eseguire lo scorrimento senza scatti pixel per pixel su EGA o VGA.*

Ridimensionamento del buffer del video

Lo scorrimento orizzontale presenta un problema. Con il modo in cui il buffer del video viene normalmente mappato, il primo byte di ogni linea di dati del buffer si trova immediatamente dopo l'ultimo byte della linea precedente. Se tentate di far scorrere l'immagine orizzontalmente con questa mappa, ogni linea sullo schermo va a capo mentre la finestra di schermo si sposta orizzontalmente nel buffer del video. Per effettuare efficacemente lo scorrimento orizzontale, dovreste ridimensionare il buffer del video in modo che ogni linea di dati del buffer sia più larga della finestra di schermo.

Il valore del registro di offset (13H) del controller del CRT controlla il modo in cui il CRTC mappa le linee nel buffer del video. Mentre scorre il quadro, il CRTC usa il valore di questo registro per localizzare l'inizio di ogni linea nell'area del buffer del video. Normalmente, le linee nel buffer del video sono della stessa larghezza delle linee visualizzate. L'aumento del valore del registro di offset allarga le linee nell'area del buffer del video facendo in modo che solo una parte di ogni linea possa essere visualizzata in un dato momento. Ciò permette di effettuare lo scorrimento orizzontale senza andare a capo.

Considerate ad esempio come potreste raddoppiare la larghezza logica del buffer del video nella modalità alfanumerica 80 per 25. Per default, il BIOS del video memorizza il valore 28H nel registro offset del CRTC in modo che il CRTC consideri ogni linea del

buffer come se fosse larga 40 word (80 byte). Sebbene ogni linea logica del buffer contenga 160 byte di dati (80 codici di carattere e 80 byte di attributo), i codici e gli attributi di carattere vengono memorizzati in diverse aree di memoria di visualizzazione (vedere Figura 99 nel Capitolo 10). Di conseguenza, per raddoppiare la larghezza logica di linea, memorizzate 50H (decimale 80) nel registro di offset del CRTC. Il CRTC visualizzerà ancora 80 caratteri in ogni riga sullo schermo, ma salterà 160 caratteri di dati tra le righe di caratteri nel buffer del video.

Quando modificate la dimensione del buffer del video programmando il registro di offset del CRTC, fate attenzione a non superare i limiti di 256 KB del buffer del video. Ad esempio, nella modalità grafica a 16 colori 640 per 350, uno schermo di pixel occupa 28.000 byte (80 byte per linea x 350 linee) in ogni area di memoria del video di 64 KB. Se ridimensionate il buffer del video aumentando il valore memorizzato nel registro di offset del CRTC, non potete oltrepassare 187 byte per linea in questa modalità video senza superare il limite di 64 KB.

La routine *BufferDims()* del Listato 121 può essere richiamata per ridimensionare il buffer del video sia nelle modalità grafiche sia in quelle alfanumeriche. Essa accetta come parametri le dimensioni orizzontali e verticali desiderate del buffer in pixel. La routine aggiorna le variabili relative nell'area dati del BIOS del video, quindi programma il registro di offset del CRTC con il valore appropriato. L'esempio del Listato 121 mostra come *BufferDims()* possa essere richiamata per trasformare una modalità alfanumerica di default di 80 per 25 in una modalità 160 per 102 nella quale possa essere utilizzata la routine *Pan()* del Listato 120.

```
                TITLE   'Listato 121'
                NAME    BufferDims
                PAGE    55,132

;
; Nome:         BufferDims
;
; Funzione:     Imposta le dimensioni del buffer del video su EGA
;
; Chiamante:    Microsoft C:
;
;                       azzera BufferDims(x,y);
;
;                                  int x,y;  /* dimensioni orizzontale */
;                                            /*  e verticale in pixel */
;
ARGx            EQU     word ptr [bp+4]
ARGy            EQU     word ptr [bp+6]

CRT_MODE        EQU     49h         ; indirizzi nell'area dati del BIOS del
                                    ;  video

CRT_COLS        EQU     4Ah
CRT_LEN         EQU     4Ch
```

```
ADDR_6845       EQU     63h
ROWS            EQU     84h
POINTS          EQU     85h

_TEXT           SEGMENT byte public 'CODE'
                ASSUME cs:_TEXT
                PUBLIC _BufferDims
_BufferDims     PROC    near

                push    bp          ; mantiene BP
                mov     bp,sp
                mov     ax,40h
                mov     es,ax       ; ES -> area dati del BIOS del video

; determina larghezza della marice di carattere visalizzato (8 o 9 pixel)

                mov     bx,8        ; BX := larga 8 pixel
                cmp     byte ptr es:[CRT_MODE],7
                                    ; verifica numero modalità BIOS
                jne     L01         ; salta se non è monocromatica
                inc     bx          ; BX := larga 9 pixel

; aggiorna area dati del BIOS del video

L01:            mov     ax,ARGx     ; AX := numero di pixel per riga
                div     bl          ; AL := numero di colonne di caratteri
                mov     es:[CRT_COLS],al
                mov     bh,al       ; BH := numero di colonne di caratteri

                mov     ax,ARGy     ; DX:AX := numero di righe di pixel
                div     byte ptr es:[POINTS]
                                    ; AL := numero di righe di caratteri
                dec     al
                mov     es:[ROWS],al

                inc     al
                mul     bh          ; AX := righe di car. * col. di carat.
                mov     es:[CRT_LEN],ax
                                    ; aggiorna area dati del BIOS del video

; aggiorna registro di offset del CRTC

                mov     ah,bh
                shr     ah,1            ; AH := numero di word per riga
                mov     al,13h          ; AL := numero registro offset del CRTC
                mov     dx,es:[ADDR_6845]     ; DX := 3B4H o 3D4H
                out     dx,ax

                pop     bp              ; ripristina BP ed esce
                ret

_BufferDims     ENDP

_TEXT           ENDS
                END
```

Listato 121. *Ridimensionamento del buffer del video.*

```
/* Listato 122 */

#define CharColumns  160        /* dimensioni carattere desiderate */
#define CharRows     102
#define CharacterWidth 8        /* 8 per modalità a colori EGA */
                                /* 9 per monocromatica EGA o VGA */
int           BytesPerRow = CharColumns * 2;
                                /* per modalità */
                                /*  alfanumeriche a 80 colonne */
main()
{
              int    i;
              int far *POINTS = 0x00400085;      /* (nell'area dati di */
                                                 /*  visualizzazione) */
              BufferDims(CharColumns*CharacterWidth,CharRows*(*POINTS));

              for( i = 0; i < CharColumns / 10; i++ )
                                /* visualizza una linea lunga */
                printf("0123456789");
              Pan( 0, 0, 80 * CharacterWidth, 0 );
                                /* scorre a destra */
              Pan( 80 * CharacterWidth, 0, 0, 0 );
                                /* scorre a sinistra */
              Pan( 0, 0, 0, 50 * (*POINTS) );
                                /* scorre verso il basso */
              Pan( 0, 50 * (*POINTS), 0, 0);
                                /* scorre verso l'alto */
}
```

Listato 122. *Creazione di una modalità alfanumerica 160 per 102.*

Stratificazione dei piani di bit

Nelle modalità grafiche a 16 colori EGA e VGA e nella modalità a 16 colori 720 per 348 della scheda InColor, potete visualizzare qualsiasi combinazione dei quattro piani di bit. Su EGA e VGA, i quattro bit meno significativi del registro di abilitazione piano colore (12H) del controller di attributo controllano quali piani di bit vengono visualizzati. Analogamente, sulla scheda InColor, i quattro bit meno significativi del registro di maschera di piano (18H) determinano quali piani di bit vengono visualizzati. In tutti i tre sistemi, tutti i quattro bit vengono impostati a 1 per abilitare la visualizzazione di tutti i quattro piani di bit. Potete azzerare qualsiasi combinazione di questi bit per impedire la visualizzazione dei piani di bit corrispondenti.

Quando disabilitate un piano di bit in questo modo, i valori di pixel vengono interpretati come se il bit corrispondente di ogni pixel fosse impostato a 0. Il contenuto di un piano di bit disabilitato non viene modificato. Questo significa che potete tracciare diverse immagini in diversi piani di bit e visualizzarle in modo selettivo. Quando i piani di bit contenenti diverse immagini vengono visualizzati contemporaneamente, le immagini sembrano sovrapporsi, come se i piani di bit fossero trasparenti e appoggiati uno sull'altro.

Considerate l'esempio della Figura 125. La griglia viene tracciata dal piano di bit 3 ed il cilindro dai piani di bit da 0 a 2 (un metodo veloce per tracciare entrambe le immagini nei piani di bit è quello di effettuare l'OR dei valori di pixel appropriati nel buffer del video). Se utilizzate una tavolozza di default a 16 colori, la griglia appare grigia e il cilindro può assumere uno dei soliti otto colori non evidenziati.

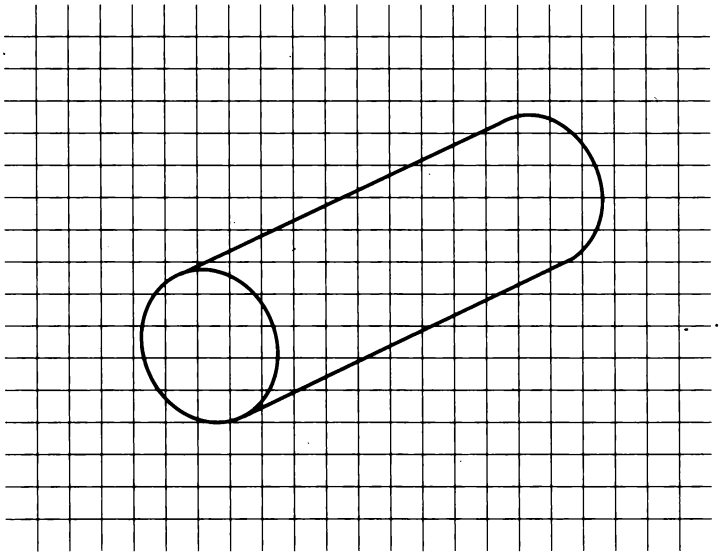


Figura 125. *Stratificazione dei piani di bit. I pixel del cilindro hanno valori tra 0 e 7 (piani di bit da 0 a 2); i pixel della griglia hanno valore 8 (solo piano di bit 3). L'abilitazione o la disabilitazione selettiva dei piani di bit da 0 a 2 e del piano di bit 3 produce la visualizzazione del cilindro, della griglia o di entrambi.*

Se venissero visualizzati tutti i quattro piani di bit, sullo schermo apparirebbero sia la griglia sia il cilindro. Se disabilitate il piano di bit 3, la griglia scompare. Se disabilitate i piani di bit da 0 a 2, visualizzando solo il piano di bit 3, scompare il cilindro e rimane visibile solo la griglia. In tutti i tre casi, il contenuto dei piani di bit resta intatto.

Nell'uso dei valori di default del registro di tavolozza con la griglia e il cilindro, scoprirete che i pixel di intersezione tra la griglia e il cilindro vengono visualizzati con colori evidenziati. Potete evitare questa situazione aggiornando la tavolozza in modo che i colori visualizzati per i punti di intersezione (i valori di pixel da 9 a 0FH) corrispondano ai colori non evidenziati (da 1 a 7). Successivamente, quando vengono visualizzati sia la griglia sia il cilindro, il cilindro appare davanti alla griglia.

Divisione schermo su EGA e VGA

Potete configurare il controller del CRT su EGA e VGA per visualizzare due diverse aree del buffer del video sullo stesso schermo (vedere Figura 126). Per realizzare questa operazione, è sufficiente programmare il registro di confronto linea (18H) del CRTC con la linea di raster in corrispondenza della quale desiderate dividere lo schermo, come mostrato nei Listati 123 e 124.

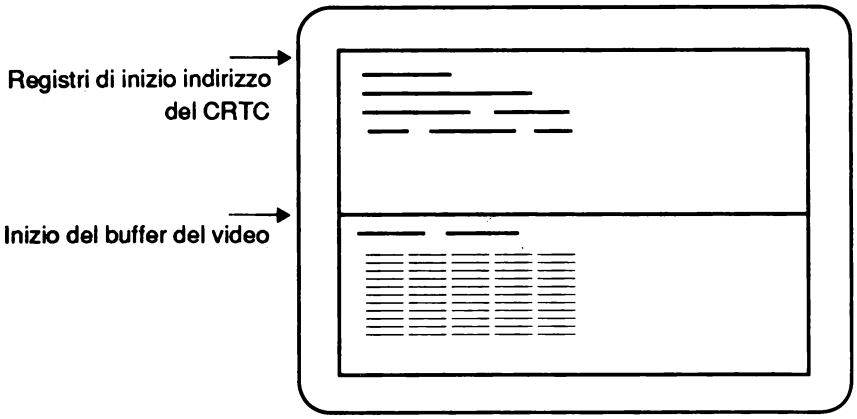


Figura 126. *Aspetto di uno schermo diviso su EGA o VGA. La parte superiore dello schermo visualizza i dati della locazione del buffer del video specificato dai registri di inizio indirizzo del CRTC. La parte inferiore dello schermo visualizza i dati dell' inizio del buffer del video.*

Il contenuto dei registri di inizio indirizzo del CRTC determinano quale area del buffer del video viene visualizzata nella parte superiore dello schermo. Mentre viene tracciato il quadro nel corso di ogni ciclo di ripristino dello schermo, il CRTC confronta la linea di scansione corrente con il valore del registro di confronto linea. Quando i valori sono uguali, il CRTC azzera il proprio contatore di indirizzo interno in modo che le restanti linee di scansione del reticolo vengano tracciate utilizzando i dati dell'inizio del buffer del video. Di conseguenza, la parte superiore del buffer del video viene sempre visualizzata nella parte inferiore dello schermo diviso.

Sia l'EGA sia la VGA possono accettare valori di confronto linea superiori a otto bit (0FFH o 256 linee di scansione) utilizzando altri registri del CRTC per contenere i bit più significativi addizionali. Di conseguenza, il bit 8 del valore di confronto linea viene rappresentato nel bit 4 del registro di superamento del CRTC (07H). Su VGA, è necessario specificare anche un nono bit per il valore di confronto linea; questo bit viene rappresentato nel bit 6 del registro di scansione linea massima (09H). La programmazione del CRTC con un valore di confronto linea richiede l'aggiornamento di due diversi registri su EGA e di tre diversi registri su VGA.

```
                TITLE   'Listato 123'
                NAME    SplitScreen
                PAGE    55,132

;
; Nome:         SplitScreen
;
; Funzione:     Divisione orizzontale dello schermo su EGA
;
; Chiamante:    Microsoft C
;
;                       azzera SplitScreen(n);
;
;                                 int n;    /* linea di scansione alla */
;                                           /*  quale dividere schermo */
;
;

ARGn            EQU     word ptr [bp+4]


ADDR_6845       EQU     63h

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC _SplitScreen
_SplitScreen    PROC    near

                push    bp              ; mantiene BP
                mov     bp,sp

                mov     ax,40h
                mov     es,ax           ; ES -> area dati del BIOS del video
                mov     dx,es:[ADDR_6845]     ; DX := porta indirizzo del CRTC

; attende ritracciamento verticale

                add     dl,6            ; DX := 3BAH o 3DAH (porta stato CRT)
L01:            in      al,dx           ; attende fine ritracciamento verticale
                test    al,8
                jnz     L01

L02:            in      al,dx           ; attende inizio ritracciam. verticale
                test    al,8
                jz      L02
                sub     dl,6            ; DX := porta indirizzo CRTC

; isola bit da 0 a 7 e bit 8 del valore di confronto linea

                mov     ax,ARGn         ; AX := valore linea di scansione
                mov     bh,ah
                and     bh,1            ; BH bit 0 := confronto linea bit 8
                mov     cl,4
                shl     bh,cl           ; BH bit 4 := confronto linea bit 8
```

```
; programma i registri del CRTC

              mov     ah,al           ; AH := 8 bit meno significc. del valore
              mov     al,18h          ; AL := numero regisro confronto linea
              out     dx,ax           ; aggiorna registro confronto linea

              mov     ah,1Fh          ; valore di default per modalità a 350
                                      ;  linee EGA (usare 11h nelle modalità
                                      ;  a 200 linee EGA)
              and     ah,11101111b    ; AH bit 4 := 0
              or      ah,bh           ; AH bit 4 := confronto linea bit 8
              mov     al,7            ; AL := numero registro di superamento
              out     dx,ax           ; aggiorna registro di superamento

              pop     bp              ; ripristina BP ed esce
              ret

_SplitScreen  ENDP

_TEXT         ENDS

              END
```

Listato 123. *Divisione dello schermo su EGA.*

```
              TITLE   'Listato 124'
              NAME    SplitScreen
              PAGE    55,132

;
; Nome:        SplitScreen
;
; Funzione:    Divisione orizzontale dello schermo su VGA
;
; Chiamante:   Microsoft C:
;
;                     azzera SplitScreen(n);
;
;                                int n;    /* linea di scansione alla */
;                                          /*  quale dividere schermo */
;
ARGn          EQU     word ptr [bp+4]


ADDR_6845     EQU     63h

_TEXT         SEGMENTbyte public 'CODE'
              ASSUME cs:_TEXT

              PUBLIC _SplitScreen

_SplitScreen  PROC    near


              push    bp              ; mantiene BP
              mov     bp,sp
```

```
            mov     ax,40h
            mov     es,ax           ; ES -> area dati del BIOS del video
            mov     dx,es:[ADDR_6845]     ; DX := porta indirizzo CRTC

; attende ritracciamento verticale

            add     dl,6            ; DX := 3BAH o 3DAH (porta stato CRT)
L01:        in      al,dx           ; attende fine ritracciamento verticale
            test    al,8
            jnz     L01

L02:        in      al,dx           ; attende inizio ritracciam. verticale
            test    al,8
            jz      L02
            sub     dl,6            ; DX := porta indirizzo CRTC

; isola bit da 0 a 7, il bit 8 e il bit 9 del valore di confronto linea

            mov     ax,ARGn         ; AX := valore linea di scansione
            mov     bh,ah           ; BH bit 0-1 := bit 8-9 del
                                    ;  valore di confronto linea

            mov     bl,bh
            and     bx,0201h        ; BH bit 1 := confronto linea bit 9
                                    ; BL bit 0 := confronto linea bit 0

            mov     cl,4
            shl     bx,cl           ; BH bit 5 := confronto linea bit 9
                                    ; BL bit 4 := confronto linea bit 8
            shl     bh,1            ; BH bit 6 := confronto linea bit 9

; aggiorna i registri del CRTC

            mov     ah,al           ; AH := 8 bit meno significa. del valore
            mov     al,18h          ; AL := numero registro confronto linea
            out     dx,ax           ; aggiorna registro confronto linea

            mov     al,7            ; AL := numero registro superamento
            out     dx,al
            inc     dx
            in      al,dx           ; AL := valore corrente reg di
                                    ;  superamento
            dec     dx

            mov     ah,al
            and     ah,11101111b    ; AH bit 4 := 0
            or      ah,bl           ; AH bit 4 := confronto linea bit 8
            mov     al,7            ; AL := numero registro superamento
            out     dx,ax           ; aggiorna registro superamento

            mov     al,9            ; AL := numero reg. linea scansione max
            out     dx,al
            inc     dx


            in      al,dx           ; AL := valore corrente reg linea
                                    ;  scansione massima
```

```
                dec     dx
                mov     ah,al
                and     ah,10111111b    ; AH bit 6 := 0
                or      ah,bh           ; AH bit 6 := confronto linea bit 9
                mov     al,9            ; AL := numero reg linea scansione
                                        ;  massima
                out     dx,ax           ; aggiorna registro linea scansione
                                        ;  massima

                pop     bp              ; ripristina BP ed esce
                ret

_SplitScreen    ENDP

_TEXT           ENDS
                END
```

Listato 124. *Divisione dello schermo su VGA.*

CONSIGLIO

Dal momento che il CRTC usa il valore di confronto linea mentre aggiorna attivamente il raster, il momento migliore per modificare questo valore è durante un intervallo di ritracciamento verticale come avviene nei Listati 123 e 124.

Il valore di default di confronto linea del BIOS del video è il valore massimo possibile (1FFH su EGA, 3FFH su VGA). Utilizzate questo valore di default per eliminare la divisione dello schermo. Esistono anche determinati valori che il CRTC non gestisce in modo utile. Sia su EGA sia su VGA, non specificate un valore di confronto linea che si trovi tra i valori di inizio ritracciamento verticale e totale verticale. Inoltre, nelle modalità a 200 linee su VGA, il valore del registro confronto linea dovrebbe corrispondere ad un numero pari.

Nelle modalità grafiche originali dell'EGA IBM, il CRTC duplica la linea di scansione alla quale lo schermo viene diviso. Questa anomalia viene anche riscontrata su alcuni cloni EGA.

Potreste trovare comodo considerare l'area inferiore dello schermo diviso come una specie di finestra sovrapposta all'area superiore. Utilizzate la prima area del buffer del video per il primo piano della finestra (la parte inferiore dello schermo diviso) ed un'altra area del buffer per lo sfondo.

Un metodo piacevole per usare la funzione di schermo diviso è quello di far scorrere lo schermo, senza scatti, verso l'alto o verso il basso. Ciò viene realizzato aumentando o diminuendo il valore del registro di confronto linea all'interno di un loop, come viene fatto dalla routine del Listato 125.

```
#define MaxScanLine  349            /* (dipende da modalità video */

for ( i = MaxScanLine; i> = 0; --i ) /* scorrimento verso l'alto */
  SplitScreen( i );

for ( i = 0; i <  MaxScanLine; i++ ) /* scorrimento verso il basso */
  SplitScreen( i );

SplitScreen( 0x3FF );               /* ripristina valore di default */
```

Listato 125. *Scorrimento verticale senza scatti di uno schermo diviso su EGA o VGA.*

L'interfaccia a penna ottica

Sulla maggior parte dei sistemi di visualizzazione trattati in questo libro, il controller del CRT può riportare la posizione della penna ottica. Quando attivate una penna ottica, essa invia un segnale al CRTC nel momento in cui il raggio elettronico dello schermo passa davanti al sensore luminoso della penna. Il CRTC risponde memorizzando il valore corrente del proprio contatore di indirizzo interno nei propri registri di penna ottica alto e penna ottica basso. Questo valore corrisponde all'offset nel buffer video dei dati visualizzati nel raster al punto in cui la penna ottica è stata attivata. Di conseguenza, il valore dei registri penna ottica alto e basso possono essere tradotti un coordinate di riga e colonna per le locazioni di schermo.

CONSIGLIO

Non è possibile collegare una penna ottica all'MDA IBM. Gli adattatori monocromatici Hercules, invece, supportano questo tipo di interfaccia. Una penna ottica usata con uno schermo monocromatico deve, tuttavia, essere in grado di operare con i fosfori ad alta persistenza P39 usati nei monitor monocromatici verdi.

Posizione della penna ottica

La posizione della penna ottica restituita dal CRTC non corrisponde ad un'esatta locazione di pixel. Uno dei motivi è semplicemente che il valore ritornato nei registri di penna ottica del CRTC è un offset di byte nel buffer del video, quindi la posizione orizzontale della penna ottica può essere determinata solo dal byte di pixel più vicino. Un'altra fonte di imprecisione è che lo stesso chip del CRTC introduce un breve ritardo tra il momento in cui riceve un segnale dalla penna ottica ed il momento in cui memorizza un valore nei propri registri di penna ottica. Il valore ritornato nei registri di penna ottica può quindi essere fino a 5 byte più grande; la quantità effettiva di errore deve essere determinata empiricamente.

L'interfaccia di programmazione della penna ottica, mostrata nella Figura 127, è analoga su tutti gli adattatori IBM e Hercules. Per determinare la posizione di una penna ottica, il vostro programma deve per prima cosa azzerare il latch di penna ottica del CRTC scrivendo uno 0 alla porta di I/O 3DBH (3BBH su MDA, adattatore Hercules o EGA con monitor monocromatico). Successivamente deve interrogare la porta di stato a 3DAH (3BAH nelle modalità monocromatiche). Quando il bit 1 del valore di porta di stato cambia da 0 a 1, la penna ottica è stata attivata e la routine può ottenere la sua posizione dal CRTC (vedere Listato 126).

Dopo aver letto la posizione della penna ottica dai registri di penna ottica, dovrete applicare una correzione empirica per il ritardo intrinseco nel CRTC. La routine del Listato 127, per la modalità alfanumerica 80 per 25 dell'EGA, sottrae 7 dal valore ritornato dal CRTC. Per convertire il risultato in una locazione di pixel, sottraete il valore dei registri di inizio indirizzo alto e inizio indirizzo basso dal valore corretto del CRTC (potete ottenere il valore di inizio indirizzo dividendo per 2 il valore di *CRT_START* nell'area dati di visualizzazione. Potete anche leggerlo dai registri di inizio indirizzo alto e inizio indirizzo basso su EGA, HGC+ e scheda InColor). Successivamente dividete la differenza per il numero di caratteri di ogni riga del buffer del video (questo valore viene rappresentato nel registro di visualizzato orizzontalmente del CRTC o in *CRT_COLS* su EGA). Il quoziente è la coordinata *y* della locazione di penna ottica. Il resto è la colonna di carattere corrispondente alla posizione della penna ottica.

MDA, HGC, HGC+ e scheda InColor	
Porta di I/O	**Funzione**
3B9H	Imposta latch di penna ottica
3BAH bit 1	Pulsante attivazione penna ottica
3BAH bit 2	Commutatore penna ottica (solo adattatori IBM)
3BBH	Azzera latch di penna ottica

CGA, EGA	
Porta di I/O	**Funzione**
3DAH bit 1	Pulsante attivazione penna ottica
3DAH bit 2	Commutatore penna ottica (solo adattatori IBM)
3DBH	Azzera latch di penna ottica
3DCH	Imposta latch di penna ottica

Figura 127. *Interfaccia di programmazione per penna ottica. Nota: nelle modalità monocromatiche EGA, leggere lo stato del pulsante di attivazione penna ottica e del commutatore da 3BAH invece che da 3DAH.*

```
                TITLE   'Listato 126'
                NAME    GetLightPen
                PAGE    55,132

;
; Nome:         GetLightPen
;
; Funzione:     Rileva posizione penna ottica
;
; Chiamante:    Microsoft C:
;
;                       int  GetLightPen();   /* ritorna offset di buffer */
;

ADDR_6845       EQU     63h

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                PUBLIC _GetLightPen
_GetLightPen    PROC    near

                push    bp
                mov     bp,sp

                mov     ax,40h
                mov     es,ax       ; ES -> area dati del BIOS del video

                mov     dx,es:[ADDR_6845]     ; DX := 3B4H o 3D4H

                add     dl,7        ; DX := 3BBH o 3DBH
                xor     al,al       ; AL := 0
                out     dx,al       ; azzera latch della penna ottica del CRTC
                jmp     $+2         ; assicura che il CRTC abbia il tempo di
                                      rispondere

                dec     dx          ; DX := 3BAH o 3DAH

L01:            in      al,dx       ; attende che la penna ottica venga
                                    ;  attivata
                test    al,2
                jz      L01

                cli                 ; disabilita interrupt
                sub     dl,6        ; DX := 3B4H o 3D4H
                mov     al,10h      ; AL := numero registro penna ottica alto
                out     dx,al
                inc     dx
                in      al,dx
                mov     ah,al       ; AH := valore penna ottica alto
                dec     dx

                mov     al,11h      ; AL := numero registro penna ottica basso
                out     dx,al
                inc     dx,
```

```
                in      al,dx       ; AX := offset al quale la penna ottica
                                    ;  è stata attivata

                sti                 ; riabilita interrupt

                pop     bp
                ret

_GetLightPen    ENDP

_TEXT           ENDS

                END
```

Listato 126. *Rilevamento della locazione della penna ottica dal CRTC.*

```
/* Listato 127 */

main()
{
                int     BufferOffset,Row,Column;
                int far*CRT_START = 0x0040004E;
                char far *CRT_COLS = 0x0040004A;

                printf( "\nCRT_COLS = %d", (int) (*CRT_COLS) );

                for( ; ; )
                {
                  BufferOffset = GetLightPen();

                  printf( "\nOffset penna ottica:  %4xh", BufferOffset );

                  BufferOffset = BufferOffset - 7; /* correzione empirica */

                  BufferOffset = BufferOffset - (*CRT_START)/2;
                                                  /* offset relat. */
                                                  /*  a inizio schermo  */

                  Row = BufferOffset / (int)(*CRT_COLS);
                                                  /* riga carattere */
                  Column = BufferOffset %  (int)(*CRT_COLS);
                                                  /* colonna carat. */

                  printf( " Colonna = %d  Riga = %d", Column, Row );
                }
}
```

Listato 127. *Uso di GetLightPen in un programma in C.*

Se pensate che sono più i problemi a cui pensare di quanto alla fine si riesca ad ottenere, avete probabilmente ragione. Sugli adattatori video IBM, oltre che nelle modalità alfanumeriche degli adattatori Hercules, potete richiamare la funzione 4 di INT 10H per ottenere la locazione di penna ottica. Se pensate di utilizzare una penna ottica nelle mo-

dalità grafiche Hercules, tuttavia, dovrete risolvere il problema facendo a meno di questa funzione.

Commutatore di penna ottica

Sugli adattatori IBM, potete determinare se il commutatore di penna ottica è stato premuto esaminando il bit 2 del valore di porta di stato restituito dalla porta 3DAH (3BAH nelle modalità monocromatiche). Questo bit viene impostato a 1 quando il commutatore è attivo e ritorna a 0 quando il commutatore è disattivato. Dovreste di norma verificare lo stato del commutatore della penna ottica prima di tentare di leggere i registri di penna ottica del CRTC.

Determinazione delle modalità video Hercules

I registri di penna ottica possono anche essere utilizzati per determinare le modalità video sugli adattatori Hercules. Nella maggior parte delle applicazioni, la determinazione della modalità video corrente non rappresenta un problema in quanto l'applicazione stessa stabilisce la modalità. Talvolta, però, un programma potrebbe non conoscere la modalità a priori. Ad esempio, un programma di stampa di schermo (vedere Appendice B) potrebbe aver bisogno di determinare la modalità video per poter interpretare correttamente il contenuto del buffer del video. Analogamente, un programma residente in RAM dovrebbe salvare e quindi ripristinare la modalità video nella quale interviene.

Potete facilmente determinare la modalità video corrente del BIOS ROM richiamando la funzione 0FH di INT 10H. Il compito risulta più difficile per gli adattatori Hercules, in quanto il BIOS non tiene presente la modalità video. Talvolta potete rilevare la modalità video dalle variabili di area dati di visualizzazione *CRT_COLS*, *CRT_LEN* e *POINTS*, ma non tutti coloro che scrivono programmi per adattatori Hercules tengono aggiornate queste variabili.

Inoltre, non esiste un metodo diretto per interrogare l'hardware per determinare la modalità video. Ad esempio, il registro di controllo modalità (3B8H), usato per selezionare la modalità video, è sfortunatamente un registro di sola scrittura. Ciononostante, potete rilevare la modalità video di un adattatore Hercules effettuando un latch dei registri di penna ottica alta e bassa (10H e 11H) del 6845 all'inizio del ritracciamento verticale, come mostrato nel Listato 128.

```
                TITLE   'Listato 128'
                NAME    GetHercMode
                PAGE    55,132

;
; Nome:         GetHercMode
;
```

```
; Funzione:   Determina modalità video su adattatori Hercules
;             valutando la dimensione
;             dell'area visualizzata del buffer del video.
;
; Chiamante:  Microsoft C:
;
;                    int GetHercMode(n);   /* ritorna dimensione */
                                           /* approssimativa */
;                                   /*  dell'area visualizzata del */
;                                   /*  buffer del video in word   */
;

_TEXT         SEGMENT byte public 'CODE'
              ASSUME cs:_TEXT

              PUBLIC _GetHercMode
_GetHercMode  PROC   near

              push   bp          ; mantiene BP
              mov    bp,sp

; azzera latch di penna ottica del CRTC

              mov    dx,3BBh     ; DX := porta di azzeramento della penna
                                 ;  ottica
              out    dx,al       ; OUT a questa porta azzera il latch
                                 ; (il valore in AL non è significativo)

; attende inizio del prossimo ritracciamento verticale

              dec    dx          ; DX := 3BAH (porta di stato del CRT)
L01:          in     al,dx       ; attende inizio del ritracciam. verticale
              test   al,80h
              jnz    L01

L02:          in     al,dx       ; attende fine del ritracciam. verticale
              test   al,80h
              jz     L02

              cli                ; disabilita interrupt
L03:          in     al,dx       ; attende inizio del ritracciam. verticale
              test   al,80h
              jnz    L03

; effettua latch del contatore corrente di indirizzo
; del CRTC nei registri di penna ottica

              dec    dx          ; DX := 3B9H
              out    dx,al       ; OUT a questa porta carica il latch
              sti                ; riabilita interrupt

; restituisce il valore nei registri di penna ottica

              mov    dl,0B4h     ; DX := 3B4H (porta di indirizzo del CRTC)
              mov    al,10h      ; AL := numero registro penna ottica alta
              out    dx,al
              inc    dx
              in     al,dx       ; legge questo registro
```

```
                dec     dx
                mov     ah,al       ; AH := valore corrente di penna ottica
                                    ;  alta

                mov     al,11h      ; AL := numero registro penna ottica bassa
                out     dx,al
                inc     dx
                in      al,dx       ; AX := valore corrente del latch di penna
                                    ;  ottica (cioè, valore del contatore di
                                    ;  indirizzo del CRTC all'inizio del
                                    ;  ritracc. verticale)
                pop     bp
                ret

_GetHercMode    ENDP

_TEXT           ENDS

                END
```

Listato 128. *Identificazione della modalità video corrente su un adattatore Hercules.*

La routine del Listato 128 attende l'inizio del ritracciamento verticale e attiva la penna ottica a questo punto con un'istruzione *OUT* alla porta 3B9H. I registri di penna ottica rispecchiano il valore del contatore di indirizzo interno del CRTC al punto in cui inizia il ritracciamento verticale (questo valore è il prodotto dei valori dei registri di visualizzato orizzontalmente e sincronizzazione verticale del CRTC). Potete supporre che i registri di penna ottica contengano almeno 7D0H (80 word per riga di carattere x 25 righe) nella modalità alfanumerica 80 per 25 e 0F4BH (45 word per riga di carattere x 87 righe) nella modalità grafica 720 per 348. L'esame del valore di penna ottica rivela così se l'HGC è in modalità alfanumerica o grafica.

CONSIGLIO

In pratica, il valore di penna ottica restituito è in qualche modo maggiore dei valori presupposti a causa del ritardo nella temporizzazione del CRTC. Questa imprecisione rende la tecnica in qualche modo meno efficace su schede HGC+ e InColor, dove dovete distinguere tra tutte le diverse dimensioni di caratteri che possono essere visualizzate dal CRTC in modalità alfanumerica. Ad esempio, il valore ritornato da *GetHercMode()* quando vengono visualizzati caratteri 9 per 8 è vicino a 0DC0H (80 x 44) e vicino a 0DB6H (90 x 39) quando vengono visualizzati caratteri 8 per 9. Dal momento che il valore di penna ottica è inesatto, potreste non essere in grado di distinguere queste due diverse configurazioni CRTC.

13
Subroutine grafiche in linguaggi ad alto livello

Linking di subroutine grafiche
Chiamate di subroutine
Interrupt ad un driver residente in memoria
Codice in linea

Aree dati globali

Interfacce grafiche stratificate
Programmazione diretta dell'hardware
Interfaccia estesa del BIOS
Interfaccia ad alto livello

La maggior parte degli esempi di programmazione del presente libro sono scritti in linguaggio Assembler, il linguaggio scelto per i programmi che devono controllare lo hardware in modo preciso e che nel contempo devono essere eseguiti il più velocemente possibile. Ciononostante, la maggior parte dei programmatori di PC IBM preferiscono non scrivere estese applicazioni interamente in linguaggio Assembler in quanto possono scrivere, mettere a punto e mantenere un programma in un linguaggio ad alto livello in modo molto più efficiente.

Mentre scrivete il codice di un programma che produce dell'output su video, dovete trovare l'equilibrio confrontando la comodità e la chiarezza concettuale fornita da un linguaggio ad alto livello e la velocità e il controllo preciso fornito da un programma in linguaggio Assembler. Una buona regola generale è quella di utilizzare il linguaggio Assembler quando accedete direttamente al buffer del video o ai registri di controllo del sistema di visualizzazione. Per tutto il resto potete in generale ottenere prestazioni soddisfacenti utilizzando un qualsiasi linguaggio ad alto livello compilato.

Questo capitolo mette a fuoco l'interfaccia tra i programmi scritti in un linguaggio ad alto livello e a basso livello e i driver in linguaggio Assembler che effettivamente accedono all'hardware del video. L'interfaccia può essere implementata in diversi modi. Il metodo da voi scelto dovrebbe dipendere dal linguaggio utilizzato, dalla vostra familiarità con i modelli di memoria e dalle tecniche di passaggio parametri impiegate dal vostro compilatore e (come sempre) dal vostro giudizio nella valutazione delle diverse alternative.

L'ultima parte di questo capitolo introduce diverse interfacce di programmazione video ad alto livello. Il punto focale è sui motivi per cui vengono utilizzate le interfacce di programmazione ad alto livello e sull'approccio di programmazione insito nel loro utilizzo.

Linking di subroutine grafiche

Potete collegare subroutine grafiche a basso livello ad applicazioni ad alto livello in diversi modi. Le tre tecniche trattate in questo capitolo (le chiamate a subroutine, le chiamate di una serie di routine residenti in memoria e l'uso di codice in linea in un programma scritto in un linguaggio ad alto livello) sono state sperimentate tutte in diverse applicazioni grafiche. Come al solito, il metodo "migliore" per usarle in una qualsiasi applicazione dipende dal proprio giudizio.

Chiamate di subroutine

Questo libro contiene numerose subroutine che sono state progettate per essere chiamate dall'interno di un programma scritto in un linguaggio ad alto livello. La maggior parte di esse devono essere collegate a programmi compilati con il compilatore Microsoft

C. Tuttavia, potete collegare queste subroutine a qualsiasi programma scritto in linguaggio ad alto livello se conoscete il corretto protocollo per la struttura del codice eseguibile e per il passaggio dei parametri ad una subroutine e il ritorno dei valori da essa. Le routine dei Listati da 129 a 136 mostrano come richiamare la stessa subroutine in linguaggio Assembler da Microsoft C, Microsoft FORTRAN, Turbo Pascal e dall'interprete BASIC.

```
                TITLE   'Listato 129'
                NAME    SetPixel
                PAGE    55,132

;
; Nome:         SetPixel
;
; Funzione:     Imposta il valore di un pixel nelle modalità grafiche EGA
;               originali.
;
; Chiamante:    Microsoft C (modello a memoria ridotta)
;
;                       azzera SetPixel(x,y,n);
;
;                       int x,y;        /* coordinate del pixel */
;
;                       int n;          /* valore del pixel */
;
; Note:         Questa è la stessa routine del Capitolo 5.
;
;

ARGx            EQU     word ptr [bp+4] ; indirizzamento della cornice
ARGy            EQU     word ptr [bp+6] ; di stack
ARGn            EQU     byte ptr [bp+8]

RMWbits         EQU     0               ; legge-modifica-scrive bit


_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr:near

                PUBLIC  _SetPixel
_SetPixel       PROC    near

                push    bp              ; mantiene registri del chiamante
                mov     bp,sp

                mov     ax,ARGy         ; AX := y
                mov     bx,ARGx         ; BX := x
                call    PixelAddr       ; AH := maschera di bit
                                        ; ES:BX -> buffer
                                        ; CL := # bit da spostare verso
                                        ;  sinistra
```

```
; imposta registro maschera di bit del controller grafico

            shl     ah,cl       ; AH := maschera di bit nella posizione
                                ;  corretta
            mov     dx,3CEh     ; porta registro di indirizzo GC
            mov     al,8        ; AL := numero registro di maschera di bit
            out     dx,ax

; imposta registro modalità controller grafico

            mov     ax,0005h    ; AL :=  numero registro modalità
                                ; AH :=  modalità di scrittura 0 (bit 0,1)
                                ;        modalità di lettura 0 (bit 3)
            out     dx,ax

; imposta il registro rotazione dati/selezione funzione

            mov     ah,RMWbits  ; AH := legge-modifica-scrive bit
            mov     al,3        ; AL := reg rotazione dati/selez. funzione
            out     dx,ax

; imposta registri impostazione/azzeramento e abilitazione
  impostazione/azzeramento

            mov     ah,ARGn     ; AH := valore di pixel
            mov     al,0        ; AL := numero reg impostazione/azzeram.
            out     dx,ax

            mov     ax,0F01h    ; AH := valore per abilit. impost./azzer.
                                ;  (tutti i piani di bit abilitati)
                                ; AL := numero reg abilit. impost./azzer.
            out     dx,ax

; imposta valore del pixel

            or      es:[bx],al  ; carica i latch durante lettura di CPU
                                ; aggiorna latch e piani di bit durante
                                ;  scrittura di CPU


; ripristina registri di default del controller grafico

            mov     ax,0FF08h   ; maschera di bit di default
            out     dx,ax

            mov     ax,0005     ; registro di modalità di default
            out     dx,ax

            mov     ax,0003     ; selezione funzione di default
            out     dx,ax

            mov     ax,0001     ; abilitaz. impostaz./azzeram. di default
            out     dx,ax
```

```
                mov     sp,bp       ; ripristina registri chiamante e ritorna
                pop     bp
                ret

_SetPixel       ENDP

_TEXT           ENDS

                END
```

Listato 129. *La subroutine SetPixel (convenzioni di richiamo per il modello a memoria ridotta del Microsoft C).*

```
/* Listato 130 */

/* traccia una rosa con n petali della forma rho = a * cos(n*theta) */

#define       Leaves            (double)11      /* n deve essere */
                                                /*  un numero dispari */

#define       Xmax              640
#define       Ymax              350
#define       PixelValue        14
#define       ScaleFactor       (double) 1.37

main()
{
              int     x,y;                  /* coordinate di pixel */
              double a;                     /* lunghezza del semiasse */
              double rho,theta;             /* coordinate polari */

              double pi = 3.14159265358979;
              double sin(),cos();

              void SetPixel();


              a = (Ymax / 2) - 1;           /* una scelta ragionevole */
                                            /*  per a */

              for (theta=0.0; theta < pi; theta+=0.001)
              {
                rho = a * cos( Leaves*theta );  /* applica la formula */

                x = rho * cos( theta );     /* converte in coordinate */
                                            /*  rettangolari */
                y = rho * sin( theta ) / ScaleFactor;

                SetPixel(x+Xmax/2,y+Ymax/2,PixelValue);
                                            /* traccia il punto */
              }
}
```

Listato 130. *Richiamo di SetPixel() da un programma in C.*

```
                TITLE   'Listato 131'
                NAME    SETPEL
                PAGE    55,132

;
; Nome:         SETPEL
;
; Funzione:     Imposta il valore di un pixel nelle modalità grafiche EGA
;               originali.
;
; Chiamante:    Microsoft Fortran
;
;                       intero*2   x,y,n
;                       richiama SETPEL(x,y,n)
;

ADDRx           EQU     dword ptr [bp+14]     ; si fa riferimento a x, y e n
ADDRy           EQU     dword ptr [bp+10]     ;  per indirizzi a 32 bit
ADDRn           EQU     dword ptr [bp+6]

RMWbits         EQU     0


SETPEL_TEXT     SEGMENTbyte public 'CODE'
                ASSUME cs:SETPEL_TEXT

                EXTRN   PixelAddr:far

                PUBLIC SETPEL
SETPEL          PROC    far

                push    bp
                mov     bp,sp

; rileva parametri tramite indirizzi a 32 bit su stack

                les     bx,ADDRn              ; ES:BX -> n
                push    es:[bx]               ; mantiene n su stack

                les     bx,ADDRy
                mov     ax,es:[bx]            ; AX := y

                les     bx,ADDRx
                mov     bx,es:[bx]            ; BX := x

                call    PixelAddr             ; calcola indirizzo di pixel

                shl     ah,cl

; programma il controller grafico

                mov     dx,3CEh
                mov     al,8
                out     dx,ax
```

```
                mov     ax,0005h
                out     dx,ax

                mov     ah,RMWbits
                mov     al,3
                out     dx,ax

                pop     ax                      ; AX := n
                mov     ah,al                   ; AH := n
                mov     al,0
                out     dx,ax

                mov     ax,0F01h
                out     dx,ax

; aggiorna il pixel, ripristina stato di default del controller grafico e
; ritorna

                or      es:[bx],al              ; aggiorna il pixel

                mov     ax,0FF08h               ; ripristina valori di default
                out     dx,ax                   ;  del controller grafico

                mov     ax,0005
                out     dx,ax

                mov     ax,0003
                out     dx,ax

                mov     ax,0001
                out     dx,ax

                mov     sp,bp                   ; ripristina registri ed esce
                pop     bp
                ret     12                      ; elimina parametri del chiamante

SETPEL          ENDP

SETPEL_TEXT     ENDS
                END
```

Listato 131. *La subroutine SETPEL (convenzioni di richiamo del Microsoft FORTRAN).*

```
c Listato 132
c traccia una rosa con n petali nella forma  rho = a * cos(n*theta)

                real*8              Leaves /11/
                real*8              ScaleFactor /1.37/

                integer*2           Xmax /640/, Ymax /350/, PixelValue /14/

                integer*2           x,y
                real*8              a
                real*8              rho,theta
```

```
            real*8              pi /3.14159265358979/
            real*8              sin,cos

            a = (Ymax/2) - 1

            do 100 theta = 0.0, pi, 0.001

            rho = a * cos( Leaves * theta )

            x = rho * cos( theta )
            y = rho * sin( theta ) / ScaleFactor

100         call SETPEL( x + Xmax/2, y + Ymax/2, PixelValue )

            stop
            end
```

Listato 132. *Richiamo di SETPEL() da un programma FORTRAN.*

```
            TITLE   'Listato 133'
            NAME    SETPEL
            PAGE    55,132

;
; Nome:       SETPEL
;
; Funzione:   Imposta il valore di un pixel in modalità a 4 colori 320x200
;
; Chiamante:  Turbo Pascal
;
;              PROCEDURE SETPEL(VAR x,y:INTEGER; PixelValue:INTEGER);
;               EXTERNAL 'setpel.bin';
;
; Note:       Il segmento di codice viene definito _TEXT in modo che
;             PixelAddr possa essere correlato allo stesso segmento
;


ADDRx       EQU     dword ptr [bp+10]     ; x e y sono variabili quindi i
ADDRy       EQU     dword ptr [bp+6]      ;  loro indirizzi vengono passati
ARGn        EQU     byte ptr [bp+4]       ; il valore di n viene passato
                                          ;  sullo stack

_TEXT       SEGMENTbyte public 'CODE'
            ASSUME cs:_TEXT

            EXTRN   PixelAddr:near

SETPEL      PROC    near

            push    bp                    ; mantiene registri del chiamante
            mov     bp,sp
            push    ds
```

```
;rende indirizzabile questa routine tramite SI

              call   L01         ; invia offset di L01

L01:          pop    si          ; CS:SI -> L01
              sub    si,offset L01

; rileva parametri tramite indirizzi a 32 bit su stack

              lds    di,ADDRy    ; DS:DI -> y
              mov    ax,[di]     ; AX := y

              lds    di,ADDRx
              mov    bx,[di]     ; BX := x

              call   PixelAddr   ; AH := maschera di bit
                                 ; ES:BX -> buffer
                                 ; CL := # di bit da spostare a sinistra

              mov    al,ARGn     ; AL := valore di pixel
              shl    ax,cl       ; AH := maschera di bit in posiz. corretta
                                 ; AH := valore di pixel in posiz. corretta

; salta alla routine corretta tramite variabile SetPixelOp

              mov    di,cs:SetPixelOp[si] ; DI := indirizzo
              add    di,si       ; DI := indirizzo riallocato
              jmp    di

                                 ; routine per sostituire valore di pixel

ReplacePixel: not    ah          ; AH := inverte maschera di bit
              and    es:[bx],ah  ; azzera il valore di pixel
              or     es:[bx],al  ; imposta il valore di pixel
              jmp    short L02

                                 ; routine per effettuare AND su valore di
                                 ;  pixel

ANDPixel:     not    ah          ; AH := inverte maschera di bit
              or     al,ah       ; AL := tutti 1 tranne valore di pixel
              and    es:[bx],al
              jmp    short L02

ORPixel:      or     es:[bx],al  ; routine per effettuare OR su valore di
                                 ;  pixel
              jmp    short L02

XORPixel:     xor    es:[bx],al  ; routine per effettuare XOR su valore di
                                 ;  pixel

L02:          pop    ds          ; ripristina registri ed esce
              mov    sp,bp
              pop    bp
              ret    10          ; elimina parametri e ritorna

SETPEL        ENDP
```

```
SetPixelOp      DW      ReplacePixel    ; contiene indirizzo operazione di pixel

_TEXT           ENDS

                END
```

Listato 133. *La subroutine SETPEL (convenzioni di chiamata Turbo Pascal).*

```
{ Listato 134 }

PROGRAM rose; { traccia una rosa con n petali della forma  rho =  a  * cos
                                                                (n*theta) }

CONST
                Leaves              = 11.0;             { deve essere un numero
                                                                  dispari }

                Xmax                = 320;
                Ymax                = 200;
                PixelValue          = 3;
                ScaleFactor         = 1.20;
                Pi                  = 3.14159265358979;

VAR
                x,y:                INTEGER;            { coordinate di pixel }
                a:                  REAL;            { lunghezza del semiasse }
                rho,theta:          REAL;                { coordinate polari }

PROCEDURE SETPEL(VAR x,y:INTEGER; PixelValue:INTEGER); EXTERNAL
'setpel.bin';

BEGIN
                GraphColorMode;       ( imposta modalità a 4 colori 320x200 }
                a := (Ymax/2) - 1;               { scelta ragionevole per a }

                theta := 0.0;
                WHILE theta < Pi DO
                BEGIN
                        rho := a * Cos(Leaves * theta);  applica la formula }

                        x := Trunc(rho*Cos(theta));        {converte in coord.
                                                                 rettangolari }
                        y := Trunc(rho * Sin(theta) / ScaleFactor);

                        x := x + Trunc(Xmax/2);          { centra su schermo }
                        y := y + Trunc(Ymax/2);

                        SETPEL(x,y,PixelValue);           { traccia il punto }
                        theta := theta + 0.001;

                END
END.
```

Listato 134. *Richiamo di SETPEL() da un programma Turbo Pascal.*

```
                TITLE   'Listato 135'
                NAME    SETPEL
                PAGE    55,132

;
; Nome:         SETPEL
;
; Funzione:     Imposta il valore di un pixel nella modalità a 4 colori 320x200
;
; Chiamante:    BASICA IBM o GWBASIC Microsoft
;
; Note:         Il segmento di codice viene definito _TEXT in modo che PixelAddr
;               possa essere
;               correlato allo stesso segmento.
;

ADDRx           EQU     word ptr [bp+10]
ADDRy           EQU     word ptr [bp+8]
ADDRn           EQU     word ptr [bp+6]

CGROUP          GROUP   _TEXT,END_TEXT

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:CGROUP,ds:CGROUP
                EXTRN  PixelAddr:near

; intestazione per BLOAD BASIC
                DB      0FDh
                DW      2 dup(0)
                DW      (offset CGROUP:BLEnd)-7 ; dimensione della subroutine

; inizio della subroutine
SETPEL          PROC    far
                push    bp          ; mantiene registri del chiamante
                mov     bp,sp
                push    es

; rende indirizzabile questa routine tramite SI

                call    L01         ; invia offset di L01
L01:            pop     si          ; CS:SI -> L01
                sub     si,offset L01

; rileva parametri tramite indirizzi a 16 bit su stack

                mov     di,ADDRy    ; DS:DI -> y
                mov     ax,[di]     ; AX := y
                mov     di,ADDRx
                mov     bx,[di]     ; BX := x
                call    PixelAddr   ; AH := maschera di bit
                                    ; ES:BX -> buffer
                                    ; CL := # di bit da spostare a sinistra
                mov     di,ADDRn
                mov     al,[di]     ; AL := valore di pixel
                shl     ax,cl       ; AH := maschera di bit in posiz. corretta
                                    ; AH := valore di pixel in posiz. corretta
```

```
; salta alla routine corretta tramite variabile SetPixelOp
              mov    di,SetPixelOp[si]     ; DI := indirizzo
              add    di,si       ; DI := indirizzo riallocato
              jmp    di
                                 ; routine per sostituire valore di pixel
ReplacePixel: not    ah          ; AH := inverte maschera di bit
              and    es:[bx],ah  ; azzera il valore di pixel
              or     es:[bx],al  ; imposta il valore di pixel
              jmp    short L02
                                 ; routine per effettuare AND su valore di
                                 ;  pixel
ANDPixel:     not    ah          ; AH := inverte maschera di bit
              or     al,ah       ; AL := tutti 1 tranne valore di pixel
              and    es:[bx],al
              jmp    short L02
ORPixel:      or     es:[bx],al  ; routine per effettuare OR su valore di
                                 ;  pixel
              jmp    short L02

XORPixel:     xor    es:[bx],al  ; routine per effettuare XOR su valore di
                                 ;  pixel

L02:          pop    es          ; ripristina registri
              mov    sp,bp
              pop    bp
              ret    6           ; elimina parametri e ritorna
SETPEL        ENDP
SetPixelOp    DW     ReplacePixel ; contiene indirizzo operazione di pixel

_TEXT         ENDS

END_TEXT      SEGMENTbyte public 'CODE'
BLEND         LABEL  BYTE        ; questo segmento viene correlato dopo
                                 ; _TEXT, quindi questa label può essere
                                 ;  usata per calcolare la dimensione del
                                 ;  segmento _TEXT
END_TEXT      ENDS
              END
```

Listato 135. *La subroutine SETPEL (convenzione di chiamata BASICA).*

```
100 ' Listato 136
110 ' Traccia una rosa con n petali della forma  rho = a * cos(n*theta)
120 DEFINT A-Z
130 LEAVES = 11
140 XMAX = 320 : YMAX = 200
150 PIXELVALUE = 2
160 SCALEFACTOR# = 1.2
170 PI# = 3.14159265358979#
180 X = 0 : Y = 0                    ' coordinate del pixel
190 A# = 0                           ' lunghezza del semiasse
200 RHO# = 0 : THETA# = 0            ' coordinate polari
210 '
220 SETPEL = 0
```

```
230 DIM SPAREA(256)                    ' riserva RAM per la subroutine
240 SETPEL = VARPTR(SPAREA(1))         ' indirizzo della subroutine
250 BLOAD "setpel.bin",SETPEL          ' carica subroutine in RAM
260 '
270 SCREEN 1 : COLOR 0,0 : CLS         ' imposta modalità a 4 colori 320x200
280 A# = (YMAX / 2) - 1                ' scelta ragionevole per A
290 THETA# = 0
300 WHILE (THETA# < PI#)
310  RHO# = A# * COS(LEAVES *
                THETA#)                ' applica la formula
320  X = RHO# * COS(THETA#)            ' converte in coordinate rettangolari
330  Y = RHO# * SIN(THETA#) / SCALEFACTOR#
340  X = X + XMAX/2                    'centra su schermo
350  Y = Y + YMAX/2
360  CALL SETPEL(X,Y,PIXELVALUE)       ' traccia il punto
370  THETA# = THETA# + .001
380  WEND
390 END
```

Listato 136. *Richiamo di SETPEL da un programma BASICA.*

Uno dei punti che maggiormente differenzia tra loro queste subroutine in linguaggio Assembler è che usano diversi modelli di memoria. Un modello di memoria descrive l'organizzazione del segmento di un programma: se il codice eseguibile è separato dai dati del programma e se ai segmenti si accede con indirizzi a 16 bit (near) o a 32 bit (far). Ad esempio, un programma a modello ridotto ha un codice "near" ed un segmento di dati "near"; un programma a modello esteso può avere un codice "far" multiplo e segmenti di dati "far". Le subroutine dei Listati da 129 a 136 sono conformi ai modelli di memoria di default usati dai diversi interpreti di linguaggio.

Anche il protocollo per il passaggio dei parametri varia a seconda dei compilatori e dei linguaggi di programmazione. Nel Pascal, ad esempio, i parametri vengono inviati allo stack nell'ordine in cui appaiono nell'istruzione *PROCEDURE*, mentre in C i parametri vengono inviati nell'ordine inverso. Inoltre, può essere inviato o il valore effettivo di un parametro o il suo indirizzo, a seconda del linguaggio di programmazione usato e del tipo di dati coinvolti. I manuali di riferimento dei compilatori contengono informazioni sul protocollo di passaggio parametri.

Microsoft C

Gli esempi in codice sorgente dei precedenti capitoli che possono essere richiamati da un programma in C sono stati tutti progettati per essere collegati con programmi a modello ridotto o compatto. Per richiamarli da un programma a modello medio o esteso, dovete apportare tre modifiche al codice sorgente per renderlo conforme alle convenzioni di chiamata subroutine di questi modelli di memoria.

- Modificate il nome del segmento di codice eseguibile.
- Usate la parola chiave *far* nelle istruzioni *PROC* dirette all'assemblatore.
- Modificate l'indirizzamento di cornice di stack in modo che si conformi all'indirizzo di ritorno a 32 bit della routine chiamante.

Ad esempio, per richiamare *SetPixel10()* in un programma in C a modello medio, modificate il nome del segmento *_TEXT* nel codice sorgente di *SetPixel10()* in un nome nel formato *modulo_TEXT* ed utilizzate la parola chiave *far* nell'istruzione *PROC* della routine. Inoltre, aumentate di due byte gli indirizzi di cornice di stack in modo che possano contenere gli indirizzi di ritorno a 32 bit.

Microsoft FORTRAN

Il compilatore FORTRAN della Microsoft non genera programmi a modello ridotto o compatto, quindi le convenzioni di indirizzamento "far" applicabili ai programmi a modelli medio ed esteso si applicano alle subroutine grafiche in linguaggio Assembler richiamabili da FORTRAN. La versione richiamabile da C del Listato 129 e l'equivalente in FORTRAN del Listato 131 si differenziano in vari modi. Queste differenze sono relative al modo in cui i parametri vengono passati attraverso lo stack alla subroutine.

Il compilatore C passa i valori correnti di ogni argomento di subroutine in ordine inverso, quindi il primo argomento viene a trovarsi in cima allo stack. Il compilatore FORTRAN passa l'indirizzo a 32 bit del valore di ogni argomento nell'ordine in cui gli argomenti appaiono nell'elenco di argomenti della subroutine. La subroutine in C ottiene i valori di argomento direttamente dallo stack; la routine FORTRAN deve ottenere gli indirizzi degli argomenti dallo stack, quindi deve usare gli indirizzi per ottenere i valori. Inoltre, in C, la routine che ha richiamato la subroutine elimina gli argomenti dallo stack. Per contro, in FORTRAN la subroutine richiamata cancella lo stack quando esce.

CONSIGLIO

I compilatori Microsoft C, FORTRAN e Pascal permettono di specificare il protocollo di passaggio parametri usato per richiamare una particolare subroutine. Ad esempio, potete scrivere una subroutine richiamabile da C e quindi accedere ad essa utilizzando la direttiva appropriata del compilatore nel vostro programma in FORTRAN o in Pascal. Questa capacità di correlazione tra i diversi linguaggi divenne possibile con la versione 3.00 dell'MS C, con la versione 3.3 dell'MS Pascal e con la versione 3.3 dell'MS FORTRAN.

L'introduzione di un'istruzione di compilazione nel vostro codice sorgente ad alto livello può rivelarsi più conveniente che modificare una subroutine in linguaggio Assembler. Ad esempio, una subroutine in C può essere richiamata da un programma in FORTRAN dichiarando la subroutine in un'unità *INTERFACE* del FORTRAN:

```
interface to subroutine SP10[C](x,y,n)
integer*2 x,y,n
end
```

Questa unità *INTERFACE* richiede al compilatore FORTRAN di generare il codice che richiama la subroutine *_sp10()* usando

il protocollo di passaggio parametri del C. Tuttavia, questa tecnica non influenza il modello di memoria utilizzato; la routine richiamabile da C viene richiamata con una chiamata "far" in quanto essa si trova in un segmento diverso dal chiamante FORTRAN. Di conseguenza, *_sp10()* deve essere sempre dichiarata con la parola chiave "far" e la cornice di stack deve essere indirizzata con l'assunto che un indirizzo di ritorno "far" a 32 bit si trova in cima allo stack quando la procedura viene richiamata

Se pensate di scrivere routine grafiche che possano essere richiamate sia da Microsoft C, sia da Pascal, sia da FORTRAN dovreste utilizzare un modello di memoria medio o esteso in modo che la routine possa essere chiamata come procedura "far". Potete utilizzare qualsiasi protocollo di passaggio parametri; gli interpreti di linguaggio Microsoft possono infatti generare codice per tutti i protocolli.

Turbo Pascal

Il Turbo Pascal collega dinamicamente le subroutine in linguaggio Assembler *EXTERNAL*. Tuttavia, il correlatore dinamico del Turbo Pascal non esegue la rilocazione di indirizzo né risolve i riferimenti simbolici tra il programma principale e la subroutine. Di conseguenza, la subroutine in linguaggio Assembler ha una struttura molto semplice. Il Listato 133 è un esempio di questo tipo di subroutine. Notare che la subroutine effettua una "auto-rilocazione" inizializzando un registro con l'offset iniziale della subroutine (usando una *CALL L01* seguita da una *POP*), aggiungendo successivamente questo valore a tutti i riferimenti alle label all'interno della subroutine.

BASIC

Il BASICA IBM e il GWBASIC Microsoft hanno proprie routine di output su video intrinseche. Tuttavia, potete utilizzare subroutine in linguaggio Assembler per personalizzare i vostri programmi in BASIC per le modalità di visualizzazione o per l'hardware del video non supportato da questi interpreti BASIC. I Listati 135 e 136 mostrano come fare.

Come il Turbo Pascal, il BASICA richiede il collegamento dinamico della subroutine. Nel Listato 135 la subroutine viene assemblata sotto forma di file binario che può essere caricato con il comando BLOAD del BASIC, come nelle righe da 220 a 250 del Listato 136. In BASICA, come in Pascal, i parametri vengono passati alla subroutine nell'ordine in cui vengono specificati nel codice sorgente ad alto livello. A differenza della subroutine in Turbo Pascal, però, la subroutine BASIC è una procedura "far". Inoltre, in BASIC gli indirizzi dei parametri vengono passati al posto dei valori dei parametri stessi.

Interrupt ad un driver residente in memoria

Un altro modo per implementare l'interfaccia tra programmi in linguaggio ad alto livello e routine grafiche in linguaggio macchina è quello di rendere residenti in memoria le routine grafiche. Quando lo sono, i programmi possono accedere alle routine grafiche eseguendo un interrupt software. Questo è il progetto che sta alla base dell'interfaccia usata da tutte le routine del BIOS del video delle famiglie di PC e di PS/2. Le routine risiedono ad un indirizzo fisso in ROM. Il vettore di interrupt 10H viene inizializzato al caricamento iniziale in modo che punti ad una routine di servizio che richiama le routine del BIOS.

Alle vostre routine di output su video si può accedere in modo analogo se vengono rese residenti in RAM e se viene impostato un vettore di interrupt che le punti (su PC e PS/2, i numeri di interrupt da 60H a 67H sono riservati a questo tipo di interrupt definiti dall'utente). Il Listato 137 è un esempio di una semplice routine residente in RAM che memorizza i pixel nella modalità a 16 colori 640 per 350 EGA. Il sorgente di questa routine viene assemblato in un file .EXE che installa la routine in RAM ed imposta il vettore di interrupt 60H in modo che punti al codice che imposta il valore di pixel. Dopo che il vettore di interrupt è stato inizializzato, qualsiasi programma può accedere alla routine caricando i registri di CPU con la locazione ed il valore di pixel e quindi eseguendo l'interrupt 60H.

```
                TITLE   'Listato 137'
                NAME    SetPixel
                PAGE    55,132

;
; Nome:         SetPixel
;
; Funzione:     Imposta il valore di un pixel nelle modalità grafiche EGA
;               originali.
;
; Chiamante:    Routine residente in memoria richiamata con interrupt 60H:
;
;                       mov ax,PixelX      ; coordinata x del pixel
;                       mov bx,PixelY      ; coordinata y del pixel
;                       mov cx,PixelValue ; valore del pixel
;
; Note:         - Assemblare e correlare per creare SETPIXEL.EXE.
;               - Eseguire una volta per rendere SetPixel residente in
                  memoria e per puntare; il vettore INT 60H al codice
                  residente in RAM.
;               -  Richiede MS-DOS versione 2.0 o successiva.
;

RMWbits         EQU     0

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT

                EXTRN   PixelAddr:near
```

```
            PUBLIC SetPixel
SetPixel    PROC   near        ; gestore di interrupt 60H residente
                               ;  in RAM
            sti                ; abilita interrupt
            push   ax          ; mantiene registri del chiamante su
            push   bx          ;  stack del chiamante
            push   cx
            push   dx
            push   cx          ; mantiene valore del pixel su stack

            call   PixelAddr   ; calcola indirizzo del pixel
            shl    ah,cl

            mov    dx,3CEh     ; programma il controller grafico
            mov    al,8        ; AL := numero registro di maschera di bit
            out    dx,ax

            mov    ax,0005h
            out    dx,ax

            mov    ah,RMWbits  ; AH := legge-modifica-scrive bit
            mov    al,3        ; AL := reg rotazione dati/selez. funzione
            out    dx,ax

            pop    ax
            mov    ah,al       ; AH := valore di pixel
            mov    al,0
            out    dx,ax

            mov    ax,0F01h
            out    dx,ax

            or     es:[bx],al  ; imposta valore di pixel

            mov    ax,0FF08h   ; ripristina valori di default del
            out    dx,ax       ;  controller grafico

            mov    ax,0005
            out    dx,ax

            mov    ax,0003
            out    dx,ax

            mov    ax,0001
            out    dx,ax

            pop    dx          ; ripristina registri chiamante e ritorna
            pop    cx
            pop    bx
            pop    ax
            iret

SetPixel    ENDP

_TEXT       ENDS
```

```
TRANSIENT_TEXT  SEGMENT  para
              ASSUME cs:TRANSIENT_TEXT,ss:STACK

Install       PROC   near

              mov    ax,2560h    ; AH := 25H (numero funzione INT 21H)
                                 ; AL := 60H (numero interrupt)
              mov    dx,seg _TEXT
              mov    ds,dx
              mov    dx,offset _TEXT:SetPixel ; DS:DX -> gestore interrupt

              int    21h         ; punta vettore INT 60H su
                                 ;  routine SetPixel

              mov    dx,cs       ; DX := segmento di inizio della parte
                                 ;  transiente
                                 ;  (eliminabile) del programma
              mov    ax,es       ; ES := prefisso del segmento del
                                 ;  programma
              sub    dx,ax       ; DX := dimensione della parte residente
                                 ;  in RAM
              mov    ax,3100h    ; AH := 31H (numero funzione INT 21H)
                                 ; AL := 0 (codice di ritorno)
              int    21h         ; conclude ma resta residente

Install       ENDP

TRANSIENT_TEXT  ENDS

STACK         SEGMENTpara stack 'STACK'

              DB     80h dup(?) ; spazio di stack per parte transiente
                                  del programma
STACK         ENDS

              END    Install
```

Listato 137. *Una routine residente in RAM per scrivere pixel in modalità grafica 640 per 350.*

Codice in linea

Una tecnica familiare a molti programmatori in C, Modula-2 e Turbo Pascal è quella di implementare subroutine a basso livello come istruzioni macchina in linea in un codice sorgente ad alto livello. Questo modo di procedere può semplificare il problema dell'uso di consistenti protocolli di modelli di memoria di passaggio parametri in quanto il compilatore di linguaggio ad alto livello gestisce tutto ciò implicitamente. Tuttavia, il codice in linea è raramente trasportabile e può essere difficilmente adattato per l'uso con altri linguaggi.

Aree dati globali

Quando correlate le subroutine di output su video ad un programma ad alto livello, dovete affrontare il problema del trasferimento delle informazioni sullo stato corrente dell'hardware del video tra il programma ad alto livello e le subroutine. Sebbene possiate passare queste informazioni alle subroutine utilizzando elenchi di argomenti, un approccio migliore è quello di utilizzare una struttura dati globale a cui possano accedere sia il programma ad alto livello sia le subroutine a basso livello. Le informazioni contenute in un'area dati globale possono comprendere:

- Indentificazione dell'hardware ("EGA con visualizzazione a colori 350 linee")
- Sistema di coordinate hardware (orientamento degli assi *x* e *y*, coordinate massime *x e y*)
- Stato del buffer del video, compreso modalità del video, dimensioni del buffer (coordinate massime *x* e *y*) e la parte di buffer visualizzata correntemente
- Valori di pixel di primo piano e di sfondo per output di testo e grafica
- Valori di colore per registri di tavolozza
- Operazione corrente di pixel (sostituzione, XOR, AND, OR o NOT)
- Modello corrente di riempimento area
- Stile di tracciamento linea corrente (linea spessa o sottile, linea tratteggiata o spezzata)

In molte applicazioni è consigliabile mantenere diverse aree globali invece di una sola. Dal momento che quasi tutti i dispositivi hardware di visualizzazione dei PC e dei PS/2 supportano più di una modalità di visualizzazione, potete creare un blocco di dati globali separato per ogni modalità e rendere "corrente" un intero blocco quando selezionate una modalità video. In un ambiente a finestre, un blocco di dati globale può applicarsi ad ogni finestra visualizzabile. Oltre alle informazioni di cui sopra, un blocco di questo tipo può anche descrivere il modo in cui le immagini grafiche ed il testo vengono mappati all'interno di una finestra. Questo può includere limiti di definizione, calcolo in scala verticale ed orizzontale o la visibilità della finestra (se una finestra è all'interno o all'esterno dello schermo, sovrapposta ad un'altra finestra e così via).

L'uso di un'area dati globale presenta diversi vantaggi. Dal momento che sia le routine ad alto livello sia quelle a basso livello possono determinare lo stato dell'hardware di output, potete scrivere programmi indipendenti dall'hardware che esaminino le informazioni descrittive nell'area dati globale per determinare come formattare l'output. Queste informazioni sono relativamente statiche, quindi mantenerle nell'area globale aiuta a ridurre i passaggi ridondanti di parametri tra le routine grafiche. Inoltre, le aree di dati globali possono essere salvate, modificate momentaneamente e ripristinate.

CONSIGLIO

Naturalmente, le informazioni nell'area dati globale possono riguardare dispositivi hardware diversi dagli adattatori video e dai monitor. Un'interfaccia grafica che supporta stampanti o plotter può anche contenere informazioni sul loro stato in un'area dati globale.

Interfacce grafiche stratificate

Dopo aver implementato un'interfaccia tra le vostre routine di output video a basso livello e i vostri programmi ad alto livello, potreste sempre scoprire che una certa quantità di codice sorgente ad alto livello riguarda manipolazioni dipendenti dall'hardware a basso livello come ad esempio il calcolo in scala delle coordinate di pixel o la definizione. E' possibile separare un codice applicativo ad alto livello da manipolazioni delle capacità hardware creando uno o più strati funzionali tra l'applicazione ad alto livello ed i driver hardware.

Nella Figura 128 è schematizzata una semplice interfaccia grafica stratificata. Lo strato inferiore comprende una serie di routine di driver hardware come quelle contenute in questo libro. Lo strato superiore fornisce una serie di subroutine che possono essere richiamate da un'applicazione ad alto livello. Le routine dello strato superiore possono richiamare direttamente i driver hardware dello strato inferiore o ci può essere uno o più strati di collegamento tra le routine ad alto livello ed i driver hardware. In ogni caso, le subroutine ad alto livello presentano una consistente interfaccia software indipendente dall'hardware per il programmatore che utilizza un linguaggio ad alto livello e di conseguenza sgravano i programmi ad alto livello dai compiti di programmazione dell'hardware del video.

Il BIOS del video ROM fornisce un esempio di questo tipo di stratificazione. La serie di routine che voi richiamate emettendo INT 10H serve da strato intermedio tra le applicazioni in linguaggio Assembler e le routine a basso livello che effettivamente programmano l'hardware. Dal punto di vista dell'applicazione, l'interfaccia INT 10H è relativamente indipendente dall'hardware; il BIOS del video programma il controller grafico, aggiorna il buffer del video ed esegue molti altri compiti di programmazione dipendenti dall'hardware. Dal momento che le routine del BIOS contengono il codice dipendente dall'hardware, un programma che impiega il BIOS è per buona parte trasportabile su diversi tipi di dispositivi hardware di visualizzazione.

Potete, naturalmente, costruire molte più funzioni all'interno di un'interfaccia stratificata di quelle fornite dal BIOS del video. Ad esempio, le interfacce grafiche di visualizzazione disponibili sul mercato possono produrre grafica sofisticata ed eseguire funzioni di controllo del video, comprese trasformazioni geometriche (calcoli in scala, traslazioni, rotazioni di immagini grafiche), grafica tridimensionale (cancellazione di linee nascoste, rappresentazione di superfici tridimensionali) o sofisticate mescolanze di colori e ombreggiature

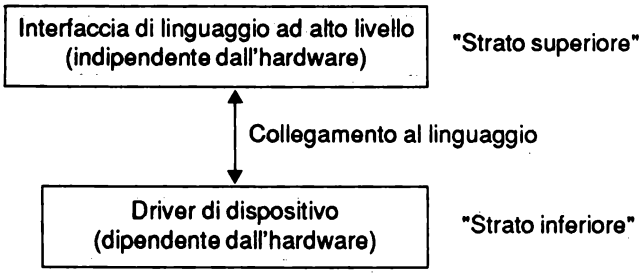


Figura 128. *Una semplice interfaccia grafica stratificata.*

Questo tipo di pacchetti grafici può supportare l'output a stampanti o plotter,oltre che ai monitor. In questo caso, l'interfaccia stratificata fornisce una serie di routine e di strutture di dati che permettono ad un programma ad alto livello di determinare lo stato di un dispositivo di output e di selezionare gli appropriati attributi di output (stile di linea, colore di tracciamento è così via) su ogni dispositivo.

In un ambiente operativo che si basa pesantemente su un'interfaccia video orientata alla grafica, l'accesso alle funzioni di sistema operativo può essere combinato con le routine di output su video in un'interfaccia di programma applicativo ad alto livello (API). Questo è l'approccio utilizzato dall'Apple Macintosh e dal Microsoft Windows. In entrambi questi ambienti, il suppporto per le funzioni di sistema come le finestre, i menu a tendina (pull-down) e le icone viene integrato in un'API unificata orientata alla grafica.

La maggior parte delle interfacce grafiche stratificate comprendono più di uno strato intermedio. Inoltre, ogni strato può essere suddiviso in diversi moduli indipendenti. La principale ragione di ciò risiede nel desiderio di mantenere la trasportabilità del software, in particolare quando il software esistente viene adattato ad un nuovo hardware di visualizzazione. Molti programmi grafici per PC sono progettati in modo che l'utente finale possa personalizzare lo strato (o gli strati) dipendente dall'hardware in base ad una particolare configurazione hardware. Questo risulta molto comodo per l'utente, dal momento che l'adattamento di un programma con un'interfaccia video stratificata ad un dispositivo hardware appena acquistato non è più difficile dell'installazione di un nuovo driver di dispositivo o di una nuova correlazione del programma con una nuova biblioteca di subroutine.

Il prezzo da pagare per questa flessibilità è una certa quantità di codice supplementare necessaria per supportare l'interfaccia stratificata, quindi i programmi operano in qualche modo più lentamente. Dovete considerare questo compromesso ogni volta che scrivete un'applicazione che si basa sull'output su schermo. Sebbene i vantaggi derivanti dall'uso di un'interfaccia grafica stratificata siano molti, molte applicazioni risultano più semplici da sviluppare ed operano più velocemente quando fate a meno dell'interfaccia grafica formale ed utilizzate solo i necessari driver a basso livello.

Per avere un'idea del tipo di programmazione necessaria quando si utilizza un'interfaccia grafica stratificata, considerate come potreste tracciare un rettangolo pieno in una modalità grafica. Gli esempi che seguono mostrano come potreste procedere in questo senso utilizzando una delle routine sviluppate precedentemente nel corso di questo libro ed impiegando due diverse interfacce grafiche stratificate. Confrontando il codice sorgente e la tecnica di programmmazione di ognuno degli esempi, noterete dove risiedono i vantaggi e gli svantaggi di ogni interfaccia grafica.

Programmazione diretta dell'hardware

La routine del Listato 138 traccia direttamente un rettangolo pieno, calcolando le estremità della serie di segmenti di linea adiacenti che costituiscono il rettangolo ed utilizzando una routine di tracciamento linee orizzontali per aggiornare il buffer del video. Questa routine potrebbe essere scritta interamente in linguaggio Assembler adattando una delle routine di tracciamento linee del Capitolo 6. La routine ad alto livello del Listato 138 opera quasi alla stessa velocità, però, dal momento che la maggior parte del tempo viene impiegata per il tracciamento delle linee, senza calcolare le loro estremità.

```
/* Listato 138 */

FilledRectangle( x1, y1, x2, y2, n )
int           x1,y1;                /* angolo superiore sinistro */
int           x2,y2;                /* angolo inferiore destro */
int           n;                    /* valore del pixel */
{
              int    y;

              for (y=y1; y<=y2; y++)
                                    /* traccia rettangolo come serie
                Line( x1, y, x2, y, n );
                                    /*  di linee orizzontali adiacenti */
}
```

Listato 138. *Uso del linguaggio C per tracciare un rettangolo pieno.*

Se la pura velocità rappresenta un argomento prioritario nel vostro programma, questo è il metodo migliore per tracciare un rettangolo. Il codice, tuttavia, è relativamente non-trasportabile in quanto presuppone implicitamente alcuni limiti dipendenti dall'hardware come ad esempio il sistema di coordinate (x,y) e le capacità di colore del sistema di visualizzazione. Non potreste utilizzare una routine come quella del Listato 138 in un ambiente operativo multitasking o a finestre in quanto essa programma direttamente l'-hardware del video e potrebbe quindi compromettere inavvertitamente l'output su video di un programma in corso di esecuzione in contemporanea.

Interfaccia estesa del BIOS

Come citato in precedenza, il BIOS ROM del video fornisce una certa quantità di indipendenza hardware e di trasportabilità grazie all'interfaccia dell'interrupt 10H. Il prezzo da pagare è naturalmente in termini di diminuzione della velocità e di mancanza di flessibilità. A prescindere da implementazioni inefficienti, le routine INT 10H sono relativamente non strutturate e hanno capacità limitate. Mano a mano che i sistemi di visualizzazione IBM sono diventati più complessi, è stata aggiunta una maggiore funzionalità all'interfaccia INT 10H, rendendola più potente ma aumentando la sua difficoltà di controllo.

Il "Direct Graphics Interface Standard (DGIS)" (standard di interfaccia grafica diretta) è un'interfaccia firmware sviluppata dalla Graphics Software Systems che aumenta le capacità dell'interfaccia INT 10H in modo strutturato. Il DGIS è stato progettato per fornire un'interfaccia uniforme a basso livello all'hardware del video sulla base di coprocessori grafici come l'Intel 82786 o il TMS34010 della Texas Instruments. La programmazione con il DGIS ricorda la programmazione con il BIOS del video dell'IBM, ma nel DGIS sono stati incorporati molti elementi di interfaccia grafica ad alto livello.

Il DGIS implementa un'interfaccia indipendente dall'hardware descrivendo gli effettivi sistemi di visualizzazione, o *dispositivi*, in termini delle loro possibili modalità di visualizzazione o *configurazioni*. Un'applicazione può interrogare il DGIS per determinare quali dispositivi sono supportati nel computer. In seguito seleziona una configurazione di output su video, sulla base della risoluzione, del numero dei colori, del supporto della grafica e/o del testo alfanumerico e così via.

Ad esempio, il Listato 139 richiama il DGIS per tracciare lo stesso rettangolo pieno visto prima. Questa volta però, invece di programmare l'hardware, il codice sorgente si occupa principalmente di programmare l'interfaccia. La routine per prima cosa stabilisce la presenza di un dispositivo di output grafico adatto nel computer richiamando la funzione di rilevamento dispositivi disponibili del DGIS. Questa funzione ritorna un elenco di dispositivi DGIS disponibili; in un sistema con un EGA, ad esempio, le configurazioni associate al dispositivo "EGA" corrispondono alle modalità video dell'EGA.

```
                TITLE   'Listato 139'
                NAME    dgisrect
                PAGE    55,132

;
; Nome:         dgisrect
;
; Funzione:     traccia un rettangolo pieno usando DGIS
;
; Note:         assemblare e correlare per creare DGISRECT.EXE
;

CR              EQU     0Dh
LF              EQU     0Ah
```

```
_TEXT           SEGMENT byte public 'CODE'
                ASSUME cs:_TEXT,ds:_DATA,ss:STACK

EntryPoint      PROC    far

                mov     ax,seg _DATA
                mov     ds,ax           ; DS ->_DATA
                push    ss
                pop     es              ; ES -> segmento di stack


; ricerca dispositivi DGIS installati

                xor     dx,dx           ; DX = 0 (lunghezza buffer)
                xor     cx,cx           ; CX = 0
                xor     bx,bx           ; BX = 0
                mov     ax,6A00h        ; AX = codice operativo DGIS (rileva
                                        ;  dispostivi disponibili)
                int     10h
                or      cx,cx
                jnz     L01             ; salta se dispositivo/i installato/i

                mov     dx,offset _DATA:Msg0
                jmp     ErrorExit

; trova dispositivo di output grafico nell'elenco dei dispositivi
;               installati DGIS

L01:            inc     cx              ; CX = (# di byte nell'elenco) + 1
                and     cx,0FFFEh       ; CX = numero pari di byte
                mov     bp,sp
                sub     sp,cx           ; stabilisce cornice di stack
                                        ;  (SS:BP -> fine cornice)
                mov     di,sp           ; ES:DI -> inizio della cornice
                                        ;  di stack

                push    di              ; salva per successivo uso
                mov     dx,cx           ; DX = dimensione del buffer
                xor     cx,cx
                xor     bx,bx
                mov     ax,6A00h        ; AX = codice operativo DGIS (rileva
                                        ;  dispositivi disponibili)
                int     10h             ; ottiene elenco dispositivi a ES:DI
                pop     di              ; ES:DI -> elenco dispositivi

L02:            cmp     word ptr
                         es:[di+2],0    ; è un dispositivo grafico?
                je      L04             ; salta in caso affermativo

                sub     bx,es:[di]      ; BX = byte restanti nell'elenco
                jnz     L03             ; salta se ci sono più dispositivi
                                        ;  nell'elenco

                mov     dx,offset _DATA:Msg1
                jmp     ErrorExit

L03:            add     di,es:[di]      ; ES:DI -> dispositivo successivo
                                        ;  nell'elenco
```

```
                jmp     L02

; stabilisce un collegamento logico con il dispositivo grafico
;  usando la prima configurazione disponibile sul dispositivo


L04:            les     di,es:[di+6]    ; ES:DI -> punto di entrata
                                        ;  dispositivo
                mov     word ptr GrDevEntry,di
                mov     word ptr GrDevEntry+2,es   ; salva punto di entrata
                mov     cx,0            ; CX = indice prima configurazione
                mov     ax,0027h        ; AX = codice operativo DGIS
                                        ;  (collegamento)
                call    dword ptr GrDevEntry ; collega a dispositivo grafico
                cmp     bx,-1           ; test dell'handle ritornato
                jne     L05             ; salta se collegato

                mov     dx,offset _DATA:Msg2
                jmp     ErrorExit

L05:            mov     ChannelHandle,bx      ; salva l'handle per uso succ.
                mov     ax,001Bh        ; AX = codice oper. DGIS
                                        ;  (inizializz. DGI)
                call    dword ptr GrDevEntry ; inizializza il dispositivo
                                             ;  con attributi di default

; traccia un rettangolo pieno usando gli attributi di default

                mov     di,100          ; DI = angolo inferiore destro y
                mov     si,100          ; SI = angolo inferiore destro x
                mov     dx,0            ; DX = angolo superiore sinistro y
                mov     cx,0            ; CX = angolo superiore sinistro x
                mov     bx,ChannelHandle      ; BX = handle
                mov     ax,003Fh        ; AX = codice operativo DGIS
                                        ;      (genera rettangolo pieno)
                call    dword ptr GrDevEntry

; scollega ed esce

                mov     bx,Channel
                        Handle          ; BX = handle
                mov     ax,002Bh        ; AX = codice operativo DGIS
                                        ;  (scollegamento)
                call    dword ptr GrDevEntry

Lexit:          mov     ax,4C00h
                int     21h             ; ritorna al DOS

ErrorExit:      mov     ah,9
                int     21h             ; visualizza messaggio errore
                mov     ax,4C01h
                int     21h             ; ritorna al DOS

EntryPoint      ENDP

_TEXT           ENDS
```

```
_DATA          SEGMENT para public 'DATA'

GrDevEntry     DD      ?          ; punto di entrata dispositivo grafico
ChannelHandle  DW      ?          ; handle per la configurazione
                                  ;  del dispositivo collegato

Msg0           DB   CR,LF,'Nessun dispositivo DGIS installato',CR,LF,'$'
Msg1           DB   CR,LF,'Nessun dispositivo grafico installato',CR,LF,'$'
Msg2           DB   CR,LF,'Impossibile collegamento a dispositivo
                       grafico',CR,LF,'$'
_DATA          ENDS

STACK          SEGMENTstack 'STACK'

               DB      400h dup(?)

STACK          ENDS

               END     EntryPoint
```

Listato 139. *Uso di DGIS per tracciare un rettangolo pieno.*

Il programma applicativo si "collega" ad una configurazione appropriata, che il DGIS identifica con un handle. L'applicazione può quindi associare un *contesto di attributo* all'handle; il contesto di attributo è una struttura di dati che definisce i colori di tracciamento, gli stili delle linee, i confini della definizione, eccetera. Chiamate successive alle funzioni di output grafico del DGIS come *OutputFilledRectangle* faranno riferimento al contesto di attributo associato ad un handle specificato.

Questa sequenza generale di operazioni è intrinsecamente flessibile. La ragione di ciò è che permette ad un programma applicativo di accedere alle funzioni hardware senza effettivamente programmare l'hardware. Ad esempio, un'applicazione può usare le funzioni DGIS per modificare una tavolozza colori o aggiornare i pixel senza scrivere direttamente sui registri di controllo hardware o sul buffer del video.

Tuttavia, un'applicazione che esegue l'output su video tramite un'interfaccia DGIS opera più lentamente di un'equivalente applicazione che programma direttamente l'hardware del video. Come sempre, quando interponete uno strato funzionale tra la vostra applicazione e l'hardware, ottenete maggiori funzionalità e trasportabilità al costo di una diminuzione di velocità. Dovete decidere se questo compromesso è accettabile o meno nelle vostre applicazioni.

Interfaccia ad alto livello

Esistono diverse implementazioni di interfaccia grafica ad alto livello disponibili per i sistemi di visualizzazione IBM. Queste interfacce ad alto livello si differenziano dal DGIS e dal BIOS del video IBM in quanto sono implementate come biblioteche software o come driver di dispositivo caricabili da RAM invece che come routine firmware. Que-

ste interfacce vi sollevano dalla necessità di programmare direttamente l'hardware e forniscono un'interfaccia di programmazione strutturata che può essere impiegata in un programma scritto in un linguaggio ad alto livello.

Le differenze tra le interfacce grafiche ad alto livello risiedono nella quantità e nel tipo di funzionalità incorporate in esse. Ad esempio, la "Virtual Device Interface" (VDI) (interfaccia di dispositivo virtuale) è una proposta di standard ANSI progettata per realizzare l'indipendenza dall'hardware dei programmi scritti in linguaggi ad alto livello. La VDI presenta un'interfaccia di programmazione compatibile con tutti i dispositivi hardware di output grafico, compresi i sistemi di visualizzazione, le stampanti e i plotter (il "Graphics Development Toolkit" venduto dalla Graphics Software Systems e dall'IBM supporta la VDI).

Un'altra diffusa interfaccia è il "Graphical Kernel System" (GKS) (sistema di base grafico), uno standard ANSI riconosciuto internazionalmente. Il GKS offre un'interfaccia altamente strutturata con potenti funzioni di manipolazione dati grafici. Il GKS non opera con dispositivi hardware individuali ma con *stazioni di lavoro* che possono comprendere diversi dispositivi di input e output collegati (come ad esempio un monitor, una tastiera ed un mouse). Un'implementazione del GKS può essere stratificata al di sopra di un'interfaccia a livello inferiore come una VDI; un'applicazione potrà quindi usare una delle interfacce senza sacrificare funzionalità o trasportabilità.

Un altro tipo ancora di interfaccia ad alto livello integra l'output grafico con l'ambiente operativo del computer, come nel caso della "Graphics Device Interface" (GDI) (interfaccia di dispositivo grafico) in Microsoft Windows. Contrariamente al DGIS, che è stato progettato per realizzare un'interfaccia a basso livello con l'hardware di visualizzazione, la GDI serve da interfaccia ad alto livello per un ambiente operativo orientato alla grafica come Windows. In un'interfaccia grafica stratificata, la GDI sarebbe più vicina allo strato superiore mentre un'interfaccia come il DGIS sarebbe più vicina allo strato inferiore. In effetti, potete installare Windows in modo che operi al di sopra del DGIS; un'applicazione di Windows potrà quindi utilizzare le funzioni GDI che a loro volta richiamano le funzioni DGIS per accedere all'hardware (Figura 129).

Il frammento di codice sorgente in C del Listato 140 si limita a scalfire la superficie della programmazione della GDI in Windows ma dovrebbe darvi un'idea di come è strutturata l'interfaccia video. La maggior parte del codice dell'esempio stabilisce un *contesto di dispositivo* affinché la funzione *Rectangle()* lo possa utilizzare. Nella GDI, un contesto di dispositivo è una struttura di dati globale che contiene informazioni sui colori con i quali vengono tracciati testo e grafica, oltre che i fattori di calcolo in scala per le coordinate (x,y) di pixel, i confini di definizione ed altre informazioni. Windows mantiene un contesto di dispositivo per ogni finestra sullo schermo. Ogni contesto di dispositivo viene identificato da un handle a 16 bit. Quando un'applicazione richiama una funzione di output della GDI come *Rectangle()* o *Ellipse()*, passa l'handle di un contesto di dispositivo alla funzione; la funzione quindi fa riferimento alle informazioni del contesto di dispositivo per produrre l'output in una finestra.

Per generare l'output grafico in una finestra, un'applicazione di Windows per prima cosa richiama la funzione di Windows *CreateWindow()*, che ritorna un handle *(hWnd)*

che identifica la finestra. L'applicazione quindi controlla la coda di messaggi di applicazione di Windows per determinare quando aggiornare la finestra.

Per generare l'output verso una finestra, l'applicazione può utilizzare un'altra funzione di Windows, *BeginPaint()*, per associare un contesto di dispositivo (identificato con l'handle *hDC*) con la finestra. L'applicazione quindi usa le funzioni GDI per stabilire gli attributi di tracciamento e la mappatura della coordinata di pixel nel contesto del disposi

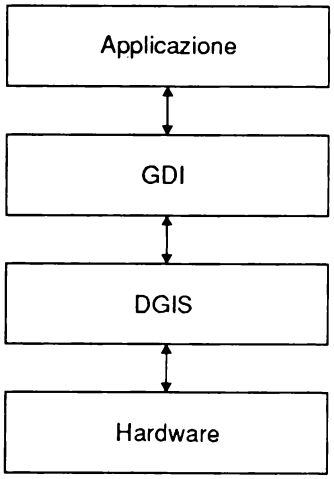


Figura 129. *GDI di Windows della Microsoft installata su DGIS.*

linea) vengono specificati creando una struttura dati che diventa parte del contesto di dispositivo.tivo. Nell'esempio del Listato 140, gli attributi della cornice del rettangolo (stile e colorelinea) vengono specificati creando una struttura dati che diventa parte del contesto di dispositivo.

```
/* Listato 140 */

    HWND hWnd;                      /* handle di finestra */
    HDC  hDC;                       /* handle del contesto di dispositivo */
    struct
    {
      HDC hdc;                      /* handle del contesto di dispositivo */
      BOOL fErase;
      RECT rcPaint;
      BOOL fRestore;
      BOOL fIncUpdate;
      BYTE rgbReserved[16];
    }                       ps;     /* "struttura di tracciamento" */


    /* crea una finestra */
```

```
hWnd = CreateWindow( ... );
.
.
.
/* inizializza contesto di dispositivo per finestra */

BeginPaint( hWnd, &ps );
hDC = ps.hdc;
.
.
.

/* associa attributi con contesto di dispositivo */

hpen = CreatePen( PS_SOLID, 0, GetSysColor(COLOR_WINDOWTEXT) );
SelectObject( hDC, (HANDLE)hpen );

hbr = CreateSolidBrush( GetNearestColor(hDC,RectFillColor) );
SelectObject( hDC, (HANDLE)hbr );
.
.
.

/* traccia un rettangolo pieno */

Rectangle( hDC, 0, 0, 100, 100 );
```

Listato 140. *Uso di GDI di Windows della Microsoft (versione 1.03) per tracciare un rettangolo pieno.*

La funzione *CreatePen()* crea la struttura dati e ritorna un handle identificativo che viene assegnato alla variabile *hpen*. La funzione *SelectObject()* aggiorna quindi il contesto di dispositivo con queste informazioni. Analogamente, le chiamate a *CreateSolidBrush()* e a *SelectObject()* stabiliscono il colore ed il modello usati per riempire il rettangolo.

Quando *Rectangle()* viene eseguito, usa gli attributi "penna" (pen) e "pennello" (brush) nel contesto di dispositivo per tracciare il contorno e l'interno del rettangolo. Le coordinate (*x*,*y*) specificate nella chiamata a *Rectangle()* indicano gli angoli superiore sinistro ed inferiore destro del rettangolo. Le coordinate non indicano locazioni assolute di pixel nel buffer del video; esse specificano punti nel sistema di coordinate che sono in relazione alla finestra nella quale viene visualizzato il rettangolo.

Il progetto generale della GDI è analogo a quello delle altre interfacce grafiche ad alto livello (le routine in linguaggio macchina dipendenti dall'hardware sono isolate nello strato inferiore dell'interfaccia, mentre le funzioni trasportabili indipendenti dall'hardware vengono implementate negli strati superiori dell'interfaccia. Le differenze tra GDI, VDI ed altre interfacce grafiche ad alto livello risiedono non tanto nei dettagli di implementazione quanto nel tipo e nella complessità delle funzioni grafiche che possono eseguire.

Appendice A
Riepilogo del BIOS del video

Tutti i computer della famiglia PC e PS/2 IBM hanno un BIOS (sistema di input/output di base) nella ROM. Il BIOS ROM contiene una serie di routine in linguaggio Assembler che forniscono un'interfaccia di programmazione a basso livello per l'accesso alle varie funzioni hardware, tra cui i drive per dischetti, il timer di sistema, le porte seriali, una stampante parallela e, naturalmente, l'hardware di visualizzazione. Costruendo un BIOS nella ROM in ogni macchina, l'IBM ha tentato di fornire un'interfaccia comune software per le varie macchine, nonostante le sostanziali differenze hardware tra i PC, i PC/XT, i PC/AT e i PS/2 IBM.

Per buona parte questa politica ha funzionato. Il trasferimento di programmi tra PC IBM con diversi dispositivi hardware tende ad essere più semplice quando i programmi accedono all'hardware solo richiamando le routine del BIOS ROM. Ciò è particolarmente vero nei programmi che manipolano lo schermo. Quando considerate le molte configurazioni video disponibili, potreste considerare il BIOS come una specie di "minimo comune denominatore" per chi sviluppa software.

Tuttavia, potreste non scegliere sempre di utilizzare le routine del BIOS ROM per l'output su video per diversi motivi. Uno di questi potrebbe essere che le routine di supporto video del BIOS ROM non sono molto veloci. Quando le prestazioni rappresentano un fattore decisivo, probabilmente vi troverete a non utilizzare queste routine. La velocità delle routine è raramente importante per i compiti svolti non di frequente, come ad esempio il caricamento di un set di caratteri in RAM o la modifica di una modalità di visualizzazione. D'altro canto, nella visualizzazione di immagini grafiche o nella produzione di effetti di animazione, l'uso del BIOS può diminuire in modo sensibile le prestazioni.

Molti altri compiti vengono eseguiti meglio dal sistema operativo piuttosto che dal BIOS. Ad esempio, quando richiamate il BIOS per scrivere caratteri sullo schermo, aggirate qualsiasi elaborazione di sistema operativo su quei caratteri. Le routine del BIOS non sanno nulla a proposito di ridirezione dell'input/output, delle finestre o di altre funzioni fornite dal sistema operativo.

Chiaramente, il BIOS ROM del video è essenziale per la programmazione video dei PC IBM, ma la misura di quanto i vostri programmi lo utilizzano è una questione di giudizio personale.

Hardware supportato dal BIOS video ROM

MDA e CGA

Il BIOS ROM sulla piastra madre di ogni PC, PC/XT e PC/AT IBM supporta sia MDA sia CGA. Inoltre, il BIOS del video del modello 30 del PS/2 supporta un MDA oltre al proprio MCGA integrato. Quando accendete un PC, il vettore dell'interrupt 10H viene inizializzato in modo che punti alla routine di servizio video in ROM.

La documentazione tecnica IBM si riferisce spesso al BIOS ROM della piastra madre dei PC e dei PS/2 come a BIOS "planare". Le routine di BIOS "planare" iniziano a F000:E000 nello spazio di indirizzo della CPU.

EGA

L'EGA IBM contiene un proprio set di driver video in ROM, situato a C000:0000. Le routine di caricamento a freddo dell'EGA inizializzano l'interrupt 10H in modo che punti la propria routine di servizio nel BIOS ROM dell'EGA. Il BIOS dell'EGA usa il vettore di interrupt 42H per puntare la routine di servizio video della piastra madre. Dal momento che le routine di interrupt 10H dell'EGA accedono alle routine del BIOS della piastra madre ogni volta che è necessario tramite INT 42H, difficilmente avrete bisogno di eseguire esplicitamente questo interrupt.

MCGA

Il BIOS ROM del video nei modelli 25 e 30 del PS/2 supporta il sottosistema MCGA integrato in questi computer. Il BIOS ROM del modello 30 supporta l'uso contemporaneo di un MDA, ma un CGA non può essere utilizzato nella stessa macchina in quanto le assegnazioni delle porte di I/O del CGA ed il suo utilizzo della memoria del video entrano in conflitto con quelli dell'MCGA.

VGA

Le routine ROM BIOS del video nei modelli 50, 60 e 80 del PS/2, a partire da E000:0000 supportano esclusivamente la VGA. Gli altri adattatori video descritti in questo libro non possono essere installati in questi computer in quanto non sono compatibili con il bus MicroChannel del PS/2.

Adattatore VGA

Le routine del BIOS ROM video dell'adattatore VGA iniziano a C000:0000. Le routine del BIOS sull'adattatore VGA sono identiche a quelle del BIOS del video dei modelli 50, 60 e 80 del PS/2, ad eccezione delle differenze secondarie relative alle diverse implementazioni hardware dell'adattatore.

Interrupt 10H

Le routine di visualizzazione del BIOS sono scritte in linguaggio Assembler e si accede ad esse eseguendo l'interrupt 10H 80x86. L'interfaccia INT 10H è progettata per i programmi in linguaggio Assembler, ma potete richiamare le routine del BIOS direttamente dai programmi scritti in linguaggi come il C o il Pascal se il vostro compilatore di linguaggio fornisce un metodo per eseguire l'interrupt.

Si seleziona la routine di supporto video del BIOS caricando un numero di funzione nel registro AH. Per passare i parametri alla routine del BIOS, introducete i loro valori nei registri 80x86 prima di eseguire INT 10H. Anche i valori che le routine del BIOS ritornano al vostro programma vengono lasciati nei registri.

Le routine del BIOS della piastra madre del PC IBM mantengono esplicitamente il contenuto dei registri DS, ES, BX, CX, DX, SI e DI (a meno che non vengano utilizzati per il passaggio dei parametri). Le routine del BIOS dell'EGA, dell'MCGA e della VGA riservano anche il registro BP.

CONSIGLIO

Se utilizzate il BIOS planare del PC o del PC/XT IBM, ricordate di riservare il registro BP tramite chiamate INT 10H al BIOS. Ad esempio:

```
push    bp          ; mantiene BP
int     10h         ; richiama il BIOS
pop     bp          ; ripristina BP
```

Come regola, le routine di input/output del video del BIOS non verificano i dati, né ritornano codici di stato o flag di errore. Di conseguenza, i vostri programmi non dovrebbero mai tentare di accedere ad un indirizzo non valido di buffer del video, selezionare una pagina video in una modalità di visualizzazione che non le supporta o accedere ad un hardware non installato nel vostro sistema. Le routine del BIOS non rilevano in modo affidabile nessuno di questi errori.

Aree dati del BIOS del video

Area dati di visualizzazione

Le routine del BIOS mantengono diverse variabili dinamiche in un'area di memoria chiamata area dati di visualizzazione. La Figura 130 contiene un riepilogo degli indirizzi di queste variabili, i loro nomi simbolici ed il loro contenuto.

Indirizzo	Nome	Tipo	Descrizione
0040:0049	CRT_MODE	Byte	Numero corrente di modalità video BIOS
0040:004A	CRT_COLS	Word	Numero di colonne di caratteri visualizzate
0040:004C	CRT_LEN	Word	Dimensione del buffer del video in byte
0040:004E	CRT_START	Word	Offset dell'inizio del buffer del video
0040:0050	CURSOR_POSN	Word	Serie di otto word contenenti la posizione del cursore per ognuna delle otto possibili pagine di video. Il byte più significativo di ogni word contiene la riga di caratteri, il byte meno significativo contiene la colonna di carattere.
0040:0060	CURSOR_MODE	Word	Linee iniziale e finale per il cursore alfanumerico. Il byte più significativo contiene la linea iniziale (superiore); il byte meno significativo contiene la linea finale (inferiore).
0040:0062	ACTIVE_PAGE	Byte	Numero di pagina video correntemente visualizzata
0040:0063	ADDR_6845	Word	Indirizzo di porta di I/O del registro di indirizzo del controller del CRT (3B4H per monocromatico, 3D4H per colori).
0040:0065	CRT_MODE_SET	Byte	Valore corrente per registro di controllo modalità (3B8H su MDA, 3D8H su CGA). Su EGA e VGA, il valore emula quelli utilizzati su MDA e CGA.
0040:0066	CRT_PALETTE	Byte	Valore corrente per registro di selezione colore (3D9H) del CGA. Su EGA e VGA il valore emula quelli utilizzati su MDA e CGA.
0040:0084	ROWS	Byte	Numero di righe di caratteri visualizzate -1
0040:0085	POINTS	Word	Altezza della matrice di carattere
0040:0087	INFO	Byte	(Vedere Figura 131)
0040:0088	INFO_3	Byte	(Vedere Figura 132)
0040:0089	Flags	Byte	Flag vari (vedere Figura 133)

(continua)

Indirizzo	Nome	Tipo	Descrizione
0040:008A	DCC	Byte	Indice tabella codice combinazioni di visualizzazione
0040:00A8	SAVE_PTR	Dword	Puntatore all'area di salvataggio del BIOS (vedere Figura 135)

Figura 130. *Area dati di visualizzazione del BIOS.*

Bit	Descrizione
7	Rispecchia il bit 7 del numero di modalità video passato alla funzione 0 di INT 10H
6-5	Quantità di RAM video: 00b - 64K 01b - 128K 10b - 192K 11b - 256K
4	(riservato)
3	1 - sistema di visualizzazione disabilitato
2	(riservato)
1	1 - sistema di visualizzazione collegato a monitor monocromatico
0	1 - emulazione cursore alfanumerico abilitata

Figura 131. *Mappatura del byte INFO a 0040:0087 nell'area dati di visualizzazione dell'EGA e della VGA.*

Bit	Descrizione
7	Input da connettore di funzione su FEAT1 (bit 6 del registro 0 di stato input) in risposta a output su FC1 (bit 1 del registro controllo funzione)
6	Input da connettore di funzione su FEAT0 (bit 5 del registro 0 di stato input) in risposta a output su FC1 (bit 1 del registro controllo funzione)
5	Input da connettore di funzione su FEAT1 (bit 6 del registro 0 di stato input) in risposta a output su FC0 (bit 0 del registro controllo funzione)
4	Input da connettore di funzione su FEAT0 (bit 5 del registro 0 di stato input) in risposta a output su FC0 (bit 0 del registro controllo funzione)
3	Commutatore di configurazione 4 (1 - off, 0 - on)
2	Commutatore di configurazione 3 (1 - off, 0 - on)
1	Commutatore di configurazione 2 (1 - off, 0 - on)
0	Commutatore di configurazione 1 (1 - off, 0 - on)

Figura 132. *Mappatura del byte INFO_3 a 0040:0088 nell'area dati di visualizzazione dell'EGA e della VGA. I bit da 4 a 7 rispecchiano lo stato di accensione del connettore di funzione. I bit da 0 a 3 rispecchiano le impostazioni dei quattro commutatori di configurazione EGA (i valori di commutatore vengono emulati dal BIOS VGA, a seconda del tipo di monitor collegato).*

Bit	Descrizione
7	Linee di scansione alfanumeriche (con bit 4):
	bit 7 bit 4
	0 0 modalità a 350 linee
	0 1 modalità a 400 linee
	1 0 modalità a 200 linee
	1 1 (riservato)
6	1 - commutazione visualizzazione abilitata
	0 - commutazione visualizzazione disabilitata
5	(riservato)
4	(vedere bit 7)
3	1 - caricamento tavolozza di default disabilitato
	0 - caricamento tavolozza di default abilitato
2	1 - uso di monitor monocromatico
	0 - uso di monitor a colori
1	1 - somma di scala di grigi abilitata
	0 - somma di scala di grigi disabilitata
0	1 - VGA abilitata
	0 - VGA disabilitata

Figura 133. *Mappatura del byte flags a 0040:0089 nell'area dati di visualizzazione dell'MCGA e della VGA.*

Le routine del BIOS del video aggiornano i valori dell'area dati di visualizzazione per rispecchiare lo stato del sistema di visualizzazione. Se modificate l'ambiente di visualizzazione senza richiamare una routine INT 10H, accertatevi di aggiornare le variabili relative nell'area dati di visualizzazione. In caso contrario si potrebbero verificare dei malfunzionamenti delle routine del BIOS video.

Aree di salvataggio

Le routine del BIOS ROM su EGA, MCGA e VGA supportano una serie di aree di salvataggio, che sono delle tabelle dinamiche di informazioni sull'hardware di visualizzazione e sul BIOS. Il BIOS del video può utilizzare queste aree di salvataggio in aggiunta all'area dati di visualizzazione. Potete anche utilizzarle per aggirare i soliti default del BIOS del video per i set di caratteri, la programmazione della tavolozza e altre funzioni di configurazione.

Le aree di salvataggio del BIOS del video sono correlate da una serie di puntatori a doppia word (segmento:offset) (vedere Figura 134). Usate la variabile SAVE_PTR (a 0040:00A8 nell'area dati di visualizzazione) per localizzare le aree di salvataggio.

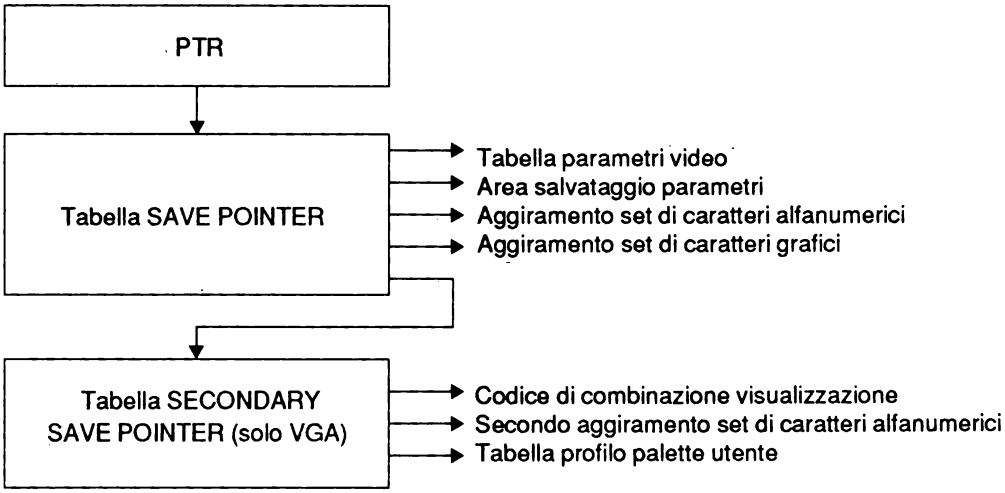


Figura 134. *Area di salvataggio del BIOS del video*

SAVE_PTR contiene l'indirizzo della tabella SAVE POINTER (vedere Figura 135). Questa tabella contiene gli indirizzi di un massimo di sette strutture di dati, ognuna delle quali presenta un diverso formato ed una diversa serie di dati relativi alle operazioni dell'hardware del video o delle routine del BIOS del video.

Il quinto indirizzo della tabella SAVE POINTER è quello della tabella SECONDARY SAVE POINTER (vedere Figura 136), che solo il BIOS VGA utilizza. Questa tabella contiene anche gli indirizzi di diverse strutture di dati il cui contenuto è relativo al funzionamento dell'hardware del video e del BIOS.

Offset	Tipo	Descrizione
0	Dword	Indirizzo della tabella dei parametri del video
4	Dword	Indirizzo dell'area di salvataggio parametri (solo EGA, VGA)
8	Dword	Indirizzo dell'aggiramento del set di caratteri alfanumerici
0CH	Dword	Indirizzo dell'aggiramento del set di caratteri grafici
10H	Dword	Indirizzo della tabella SECONDARY SAVE POINTER (solo VGA)
14H	Dword	(riservato)
18H	Dword	(riservato)

Figura 135. *Tabella SAVE POINTER (EGA, MCGA, VGA).*

Offset	Tipo	Descrizione
0	Word	Lunghezza della tabella SECONDARY SAVE POINTER in byte
2	Dword	Indirizzo della tabella codice combinazioni visualizzazioni
6	Dword	Indirizzo del secondo aggiramento del set di caratteri alfanumerici
0AH	Dword	Indirizzo della tabella profilo tavolozza utente
0EH	Dword	(riservato)
12H	Dword	(riservato)
16H	Dword	(riservato)

Figura 136. *Tabella SECONDARY SAVE POINTER (solo VGA).*

A parte le tabelle SAVE POINTER e SECONDARY SAVE POINTER, le uniche strutture di dati fornite dal BIOS ROM sono la tabella di parametri video e, su VGA, la tabella di codice combinazioni di visualizzazione. Di conseguenza, i soli indirizzi inizializzati della tabella SAVE POINTER sono quelli della tabella dei parametri video e della tabella SECONDARY SAVE POINTER. Il solo indirizzo inizializzato nella tabella SECONDARY SAVE POINTER appartiene alla tabella codice combinazioni di visualizzazione. I restanti indirizzi vengono inizializzati a 0.

Tabella parametri video

Questa struttura dati contiene parametri di configurazione utilizzati dalle routine di impostazione modalità di visualizzazione BIOS video. La tabella contiene elementi per ogni modalità video disponibile. La sua struttura si differenzia sui vari modelli dell'EGA, dell'MCGA e della VGA. La Figura 137 rappresenta un elemento tipico nella tabella di parametri video della VGA. I formati per gli elementi di tabella nel BIOS dell'EGA e dell'MCGA sono simili.

Offset	Tipo	Descrizione
0	Byte	Valore per CRT_COLS
1	Byte	Valore per ROWS
2	Byte	Valore per POINTS
3	Word	Valore per CRT_LEN
5	Serie di 4 byte	Valori per registri del sequencer da 1 a 4
9	Byte	Valore per registro output varie

(continua)

Offset	Tipo	Descrizione
0AH	Serie di 25 byte	Valori per registri del CRTC da 0 a 18H
23H	Serie di 20 byte	Valori per registri del controller di attributo da 0 a 13H
37H	Serie di 9 byte	Valori per registri del controller grafico da 0 a 8

Figura 137. *Formato di un elemento di tabella parametri video VGA. La tabella di parametri video VGA comprende 29 elementi di questo tipo.*

Area salvataggio parametri

Quando è presente, questa tabella contiene i valori dei registri di tavolozza (da 00H a 0FH) del controller grafico dell'EGA o della VGA e del registro di sovrascansione (11H), come mostrato nella Figura 138. Il BIOS del video aggiorna l'area di salvataggio parametri ogni volta che aggiorna i corrispondenti registri del controller di attributo.

Offset	Tipo	Descrizione
0	Serie di 16 byte	Contenuto corrente dei registri di tavolozza del controller grafico
10H	Byte	Contenuto corrente registro di sovrascansione del controller grafico
11H-0FFH	(riservato)	

Figura 138. *Area salvataggio parametri. Questa area ha una dimensione di 256 byte.*

CONSIGLIO

Quando un profilo tavolozza utente (vedere Figura 142 più avanti in questa spiegazione) si sovrappone ai valori di default del registro di tavolozza, l'area di salvataggio parametri viene aggiornata con i valori di default e non con quelli del profilo tavolozza utente.

Aggiramento del set di caratteri alfanumerici

Questa struttura dati (vedere Figura 139) indica un set di caratteri alfanumerici da impiegare al posto del set di caratteri di default del BIOS. Il set di caratteri viene caricato ogni volta che il BIOS del video viene richiamato per selezionare una delle modalità video specificate dalla struttura dati.

Offset	Tipo	Descrizione
0	Byte	Lunghezza in byte di ogni definizione carattere
1	Byte	Banco di RAM del generatore di caratteri
2	Word	Numero di caratteri definiti
4	Word	Primo codice di carattere nella tabella
6	Dword	Indirizzo della tabella di definizione caratteri
0AH	Byte	Numero delle righe di caratteri visualizzate
0BH	Serie di byte	Modalità video applicabili
	Byte	0FFH (fine dell'elenco delle modalità video)

Figura 139. *Aggiramento del set di caratteri alfanumerici.*

Su VGA, potete specificare un secondo set di 256 caratteri creando una seconda struttura dati di aggiramento set di caratteri alfanumerici e memorizzando il suo indirizzo nella tabella SECONDARY SAVE POINTER.

Aggiramento set di caratteri grafici

Questa struttura dati (vedere Figura 140) si sovrappone alla selezione di set di caratteri di default del BIOS ogni volta che il BIOS del video imposta una delle modalità di visualizzazione specificate.

Offset	Tipo	Descrizione
0	Byte	Numero di righe di caratteri visualizzate
1	Word	Lunghezza in byte di ogni definizione carattere
3	Dword	Indirizzo della tabella di definizione caratteri
7	Serie di byte	Modalità video applicabili
	Byte	0FFH (fine dell'elenco delle modalità video)

Figura 140. *Aggiramento del set di caratteri grafici.*

Tabella di codice combinazioni di visualizzazione

La Figura 141 elenca tutte le combinazioni dei sistemi di visualizzazione supportati dal BIOS del video. La descrizione della funzione 1AH di INT 10H in questa appendice spiega come viene utilizzata questa tabella

CONSIGLIO

Il BIOS del video dell'MCGA contiene una tabella di codice di combinazioni visualizzazioni in ROM per supportare la funzione 1AH di INT 10H. Tuttavia, il BIOS dell'MCGA non supporta una tabella SECONDARY SAVE POINTER, quindi non potete modificare la sua tabella DCC.

Offset	Tipo	Descrizione
0	Byte	Numero di elementi nella tabella
1	Byte	Numero versione della tabella DCC
2	Byte	Codice del tipo di visualizzazione massimo
3	Byte	(riservato)
4	Serie di word	Ogni coppia di byte nella serie descrive una combinazione valida (vedere la funzione 1AH di INT 10H)

Figura 141. *Tabella di codice di combinazioni di visualizzazione.*

Tabella profilo tavolozza utente

Questa struttura dati contiene aggiramenti specificati dall'utente per i valori di default dei registri di sovrascansione e di tavolozza del controller di attributo per i valori di default dei 256 registri di colore del DAC del video, e per il valore di default del registro di locazione sottolineatura del CRTC (vedere Figura 142). Solo il BIOS del video della VGA supporta questa tabella.

Offset	Tipo	Descrizionel		
0	Byte	Sottolineatura:	1	Abilitata in tutte le modalità alfanumeriche
			0	Abilitata nella modalità alfanumerica monocromatica
			- 1	Disabilitata in tutte le modalità alfanumeriche
1	Byte	(riservato)		
2	Word	(riservato)		
4	Word	Numero di registri del controller di attributo nella tabella		
6	Word	Numero del primo registro del controller di attributo		
8	Dword	Indirizzo della tabella dei registri del controller di attributo		
0CH	Word	Numero di registri di colore del DAC del video nella tabella		
0EH	Word	Numero del primo registro di colore del DAC del video		
10H	Dword	Indirizzo della tabella dei registri di colore del DAC del video		
14H	Serie di byte	Modalità video applicabili		
	Byte	0FFH (fine dell'elenco delle modalità video)		

Figura 142. *Tabella profilo tavolozza utente.*

Programmazione dell'area di salvataggio del BIOS del video

Per utilizzare una struttura dati supportata nelle tabelle SAVE POINTER e SECONDARY SAVE POINTER, basta introdurre la struttura dati in RAM ed aggiornare gli appropriati indirizzi di SAVE POINTER e di SECONDARY SAVE POINTER, in modo che venga puntata da essi. Dal momento che le tabelle di default di SAVE POINTER e di SECONDARY SAVE POINTER sono allocate in ROM, dovete copiare queste tabelle in RAM ed aggiornare SAVE_PTR (0040:00A8) in modo appropriato prima di modificarle.

I Listati 141 e 142 mostrano due usi delle aree di salvataggio del BIOS. La routine del Listato 141 fornisce un'area di salvataggio parametri per il BIOS EGA o VGA. Una volta stabilita l'area di salvataggio parametri, i suoi primi 17 byte vengono aggiornati con il contenuto dei 16 registri di tavolozza del controller di attributo e del suo registro di sovrascansione ogni volta che il BIOS del video scrive su di essi.

```
                TITLE   'Listato 141'
                NAME    EstablishPSA
                PAGE    55,132

;
; Nome:         EstablishPSA
;
; Funzione:     Stabilisce un'area di salvataggio parametri per il BIOS del
;               video EGA o VGA.
;               Questa area di salvataggio rispecchierà i valori correnti
;               dei registri di sovrascansione e di tavolozza del controller
;               di attributo.
;
; Chiamante:    Microsoft C:
;
;                       azzera EstablishPSA()
;

SAVE_PTR        EQU     0A8h

DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP,es:DGROUP

                PUBLIC _EstablishPSA
_EstablishPSA PROC      near

                push    bp
                mov     bp,sp
                push    si
                push    di

; mantiene precedente SAVE_PTR

                push    ds
```

```
            pop     es          ; ES -> DGROUP
            mov     di,offset DGROUP:Old_SAVE_PTR

            mov     ax,40h
            mov     ds,ax       ; DS -> area dati del BIOS del video
            mov     si,SAVE_PTR; DS:SI -> SAVE_PTR

            mov     cx,4
            rep     movsb

; copia tabella SAVE POINTER in RAM

            lds     si,es:Old_SAVE_PTR    ; DS:SI -> tabella SAVE POINTER
            mov     di,offset DGROUP:SP_TABLE1
            mov     cx,7*4      ; numero di byte da spostare
            rep     movsb

; aggiorna SAVE_PTR con indirizzo della nuova tabella SAVE POINTER

            mov     ds,ax       ; DS -> area dati del BIOS del video
            mov     si,SAVE_PTR
            mov     word ptr [si],offset DGROUP:SP_TABLE1
            mov     [si+2],es

; aggiorna tabella SAVE POINTER con indirizzo dell'area di salvataggio
; parametri

            push    es
            pop     ds          ; DS -> DGROUP

            mov     word ptr SP_TABLE1[4],offset DGROUP:PSA
            mov     word ptr SP_TABLE1[6],ds

; ripristina registri ed esce

            pop     di
            pop     si
            mov     sp,bp
            pop     bp
            ret

_EstablishPSA ENDP

_TEXT         ENDS


_DATA         SEGMENTword public 'DATA'

Old_SAVE_PTR  DD      ?           ; precedente valore di SAVE_PTR

SP_TABLE1     DD      7 dup(?)    ; copia RAM della tabella SAVE POINTER
```

```
PSA             DB      256 dup(0) ; area di salvataggio parametri

_DATA           ENDS

                END
```

Listato 141. *Uso di un'area di salvataggio parametri per controllare i registri di tavolozza dell'EGA e della VGA.*

Il Listato 142 mostra come specificare i valori della tavolozza da utilizzarsi quando le routine del BIOS del video vengono richiamate per stabilire una nuova modalità video. Per prima cosa, introdurre i valori in una tabella il cui indirizzo è memorizzato in una struttura dati di profilo tavolozza utente. Successivamente, introdurre l'indirizzo di questa struttura dati nella tabella SECONDARY SAVE POINTER (dal momento che questo esempio usa la tabella SECONDARY SAVE POINTER, potete eseguirlo solo su VGA).

```
                TITLE   'Listato 142'
                NAME    EstablishUPP
                PAGE    55,132

;
; Nome:         EstablishUPP
;
; Funzione:     Stabilisce un'area di salvataggio profilo tavolozza utente
;               per il BIOS del video VGA.
;               Questa area di salvataggio sovrappone ai soliti valori di
;               default di palette un elenco specificato di modalità video.
;
; Chiamante:    Microsoft C:
;
;                       azzera EstablishUPP();
;

SAVE_PTR        EQU     0A8h

DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP,es:DGROUP

                PUBLIC _EstablishUPP
_EstablishUPP   PROC    near

                push    bp
                mov     bp,sp
                push    si
                push    di

; mantiene precedente SAVE_PTR

                push    ds
                pop     es          ; ES -> DGROUP
```

```
            mov     di,offset DGROUP:Old_SAVE_PTR

            mov     ax,40h
            mov     ds,ax       ; DS -> area dati del BIOS del video
            mov     si,SAVE_PTR; DS:SI -> SAVE_PTR

            mov     cx,4
            rep     movsb

; copia tabella SAVE POINTER in RAM

            lds     si,es:Old_SAVE_PTR   ; DS:SI -> tabella SAVE POINTER
            mov     di,offset DGROUP:SP_TABLE1
            mov     cx,7*4      ; numero di byte da spostare
            rep     movsb

; aggiorna SAVE_PTR con l'indirizzo della nuova tabella SAVE POINTER

            mov     ds,ax       ; DS -> area dati del BIOS del video
            mov     si,SAVE_PTR
            mov     word ptr [si],offset DGROUP:SP_TABLE1
            mov     [si+2],es

; copia tabella SECONDARY SAVE POINTER in RAM

            lds     si,es:SP_TABLE1[16]  ; DS:SI -> tabella SEC SAVE POINTER
            mov     di,offset DGROUP:SP_TABLE2
            mov     cx,[si]
            rep     movsb

; aggiorna nuova tabella SAVE POINTER con indirizzo della nuova tabella
; SECONDARY SAVE POINTER

            push    es
            pop     ds          ; DS -> DGROUP

            mov     word ptr SP_TABLE1[16],offset DGROUP:SP_TABLE2
            mov     word ptr SP_TABLE1[18],ds

; aggiorna SECONDARY SAVE POINTER con indirizzo del profilo tavolozza
; utente

            mov     word ptr SP_TABLE2[10],offset DGROUP:UPP
            mov     word ptr SP_TABLE2[12],ds

; ripristina registri ed esce

            pop     di
            pop     si
            mov     sp,bp
            pop     bp
            ret

_EstablishUPP ENDP

_TEXT         ENDS
```

```
_DATA          SEGMENTword public 'DATA'

Old_SAVE_PTR   DD      ?           ; valore precedente di SAVE_PTR

SP_TABLE1      DD      7 dup(?)    ; copia della tabella SAVE POINTER

SP_TABLE2      DW      ?           ; copia della tabella SECONDARY SAVE
                                   ;  POINTER
               DD      6 dup(?)

UPP            DB      0           ; flag sottolineatura
               DB      0           ; (riservato)
               DW      0           ; (riservato)
               DW      17          ; # di registri di sovrascansione e di
                                   ;  tavolozza
               DW      0           ; primo registro specificato nella tabella
               DW      DGROUP:PalTable; puntatore alla tabella di tavolozza
               DW      seg DGROUP
               DW      0           ; numero di reg colore del DAC del video
               DW      0           ; primo registro del DAC del video
               DD      0           ; puntatore alla tabella colori del DAC
                                   ;  del video
               DB      3,0FFh      ; elenco delle modalità video applicabili

PalTable       DB      30h,31h,32h,33h,34h,35h,36h,37h; una tavolozza
               DB      00h,01h,02h,03h,04h,05h,14h,07h; personalizzata
               DB      01h             ; reg di sovrascansione

_DATA          ENDS

               END
```

Listato 142. *Uso di un profilo palette utente per aggirare la tavolozza di default VGA.*

CONSIGLIO

In generale, la vostra applicazione dovrebbe riportare al suo valore originale SAVE_PTR quando le tabelle SAVE POINTER e le aree di salvataggio non sono più necessarie. Se desiderate mantenere queste tabelle in RAM per farle utilizzare a successive applicazioni, usate la funzione "Concludi ma resta residente" dell'MS-DOS (funzione 31H di INT 21H) in modo che la RAM contenente le tabelle non venga liberata quando il programma che le ha create conclude le sue operazioni.

Vettore di interrupt 1DH

Questo vettore di interrupt contiene l'indirizzo di una tabella di valori di inizializzazione video (vedere Figura 143). Questi valori sono utili solo per MDA e CGA; tuttavia, la tabella viene mantenuta per compatibilità tra tutti i PC e i PS/2.

Offset	Tipo	Descrizione
0	Serie di 16 byte	Registri del CRTC per modalità alfanumerica 40x25 (CGA)
10H	Serie di 16 byte	Registri del CRTC per modalità alfanumerica 80x25 (CGA)
20H	Serie di 16 byte	Registri del CRTC per modalità grafiche a 4 colori 320x200 o a 2 colori 640x200 (CGA)
30H	Serie di 16 byte	Registri del CRTC per monocromatica 80x25 (MDA)
40H	Word	Lunghezza del buffer del video (modalità alfanumerica 40x25)
42H	Word	Lunghezza del buffer del video (modalità alfanumerica 80x25)
44H	Word	Lunghezza del buffer del video (modalità grafiche CGA)
46H	Word	Lunghezza del buffer del video (modalità grafiche CGA)
48H	Serie di 8 byte	Numero delle colonne di caratteri visualizzati per le modalità del BIOS del video da 0 a 7
50H	Serie di 8 byte	Valori per il registro di controllo modalità del CRT 3x8H per le modalità del BIOS del video da 0 a 7

Figura 143. *Tabella di inizializzazione video MDA e CGA. L'indirizzo di questa tabella è memorizzato nel vettore per INT 1DH.*

Funzioni del BIOS del video dei PC e dei PS/2 IBM (interfaccia INT 10H)

Le pagine che seguono forniscono descrizioni dettagliate di ogni funzione BIOS disponibile tramite interrupt software 10H. Le descrizioni hanno lo scopo di completare i riepiloghi delle funzioni e i listati di codice sorgente in linguaggio Assembler della documentazione tecnica IBM. I frammenti di codice sorgente inclusi rappresentano esempi tipici di programmazione che potrete modificare per adattarli ai vostri scopi.

Questo riepilogo comprende informazioni sulle routine del BIOS ROM situato sulla piastra madre, sull'EGA, sull'MCGA e sulla VGA. Tuttavia, non tutte le routine sono disponibili o funzionano in modo identico su tutti i computer della famiglia dei PC e dei PS/2 IBM.

Tutte le informazioni di questo capitolo sono basate sulle specifiche tecniche IBM e sulle successive versioni della ROM del video distribuite alla data indicata:

- ROM della piastra madre del PC IBM: 27/10/82
- ROM della piastra madre del PC/AT IBM: 10/6/85

- ROM EGA IBM: 13/9/84
- ROM (MCGA) del modello 30 del PS/2 IBM: 2/9/86
- ROM (VGA) del modello 60 del PS/2 IBM: 13/2/87
- ROM dell'adattatore video (VGA) del PS/2 IBM: 27/10/86

Funzione 0: selezione modalità video

Registri del chiamante:

AH = 0

AL = numero di modalità video:

0	alfanumerica a 16 colori 40 per 25, burst di colore disabilitato
1	alfanumerica a 16 colori 40 per 25, burst di colore abilitato
2	alfanumerica a 16 colori 80 per 25, burst di colore disabilitato
3	alfanumerica a 16 colori 80 per 25, burst di colore abilitato
4	grafica a 4 colori 320 per 200, burst di colore abilitato
5	grafica a 4 colori 320 per 200, burst di colore disabilitato
6	grafica a 2 colori 640 per 200, burst di colore abilitato
7	alfanumerica monocromatica 80 per 25 (solo MDA, Hercules, EGA e VGA)
8	grafica a 16 colori 160 per 200 (solo PCjr)
9	grafica a 16 colori 320 per 200 (solo PCjr)
0AH	grafica a 4 colori 640 per 200 (solo PCjr)
0BH	riservato (usato dalla funzione 11H del BIOS EGA)
0CH	riservato (usato dalla funzione 11H del BIOS EGA)
0DH	grafica a 16 colori 320 per 200 (solo EGA e VGA)
0EH	grafica a 16 colori 640 per 200 (solo EGA e VGA)
0FH	grafica monocromatica 640 per 350 (solo EGA e VGA)
10H	grafica a 16 colori 640 per 350 (VGA, EGA con almeno 128 KB)
	grafica a 4 colori 640 per 350 (EGA con 64 KB)
11H	grafica a 2 colori 640 per 480 (solo MCGA, VGA)
12H	grafica a 16 colori 640 per 480 (solo VGA)
13H	grafica a 256 colori 320 per 200 (solo MCGA e VGA)

Valori restituiti:

(nessuno)

Aggiornamenti area dati di visualizzazione:

0040:0049	CRT_MODE
0040:004A	CRT_COLS
0040:004C	CRT_LEN

0040:004E CRT_START
0040:0050 CURSOR_POSN
0040:0060 CURSOR_MODE
0040:0062 ACTIVE_PAGE
0040:0063 ADDR_6845
0040:0065 CRT_MODE_SET
0040:0066 CRT_PALETTE
0040:0084 ROWS
0040:0085 POINTS
0040:0087 INFO
0040:0088 INFO_3

La funzione 0 di INT 10H attiva nel sistema di visualizzazione la modalità video da voi specificata con il valore memorizzato nel registro AL. La funzione 0 programma il controller del CRT, seleziona una tavolozza di colori di default ed opzionalmente cancella il buffer del video. Potete modificare diversi compiti di default eseguiti dalla funzione 0 impostando i flag nell'area dati di visualizzazione (vedere funzione 12H di INT 10H) o fornendo un aggiramento del set di caratteri o di attributo di tavolozza nelle aree di salvataggio del BIOS.

I numeri di modalità video 0BH e 0CH sono riservati alla routine di supporto del BIOS EGA per i set di caratteri caricabili da RAM, nei quali la mappa di memoria video 2 viene selettivamente abilitata in modo che possa essere caricata una tabella di definizioni caratteri.

Su EGA, MCGA e VGA, i monitor compositi non sono supportati e non esiste un segnale di burst di colore da controllare. Di conseguenza, la modalità 0 è identica alla modalità 1, la modalità 2 è uguale alla modalità 3 e la modalità 4 alla modalità 5.

Se usate questa routine del BIOS per richiedere una modalità video non supportata dal vostro hardware di sistema, i risultati sono imprevedibili. In particolare, se selezionate la modalità 7 (alfanumerica monocromatica) con un CGA, il BIOS della piastra madre programma il CRTC del CGA con parametri appropriati per MDA, cosa che produce un disturbo incomprensibile sul monitor CGA. Il terzo esempio riportato qui di seguito mostra come risolvere questo problema impostando i bit 4 e 5 di EQUIP_FLAG (0040:0010) in modo da indicare quale sistema di visualizzazione deve essere utilizzato dal BIOS.

Su EGA, MCGA e VGA, se il bit 7 del numero di modalità video richiesta in AL è impostato a 1, il buffer del video non viene cancellato quando viene selezionata una nuova modalità video. Di conseguenza, un programma può passare alternativamente da un sistema di visualizzazione ad un altro senza perdere il contenuto dei rispettivi buffer del video.

Il seguente esempio seleziona la modalità grafica a 4 colori 320 per 200.

```
mov             ax,0004             ; AH := 0 (numero di funzione INT 10H)
                                    ; AL := 4 (numero di modalità video)
int             10h
```

Questa routine mostra come cambiare modalità su EGA senza cancellare il buffer del video.

```
mov             ax,000EH            ; seleziona una modalità video (in questo
                                    ;  caso, la modalità a 16 colori 640x200)
or              al,10000000b        ; imposta bit 7
int             10h
```

Per selezionare le modalità video in un sistema contenente sia un CGA sia un MDA, usate una routine come la seguente:

```
mov             ax,40h
mov             es,ax
and             byte ptr es:[10h],11001111b ; azzera bit 4 e 5 di EQUIP_FLAG
or              byte ptr es:[10h],00110000b ; imposta bit 4 e 5:
                                            ;  11b - monocromatica
                                            ;  10b - a colori (80x25)
                                            ;  01b - a colori (40x25)
                                            ;  00b - (non usato)
mov             ax,0007
int             10h                         ; seleziona modalità
                                            ;  monocromatica 7

and             byte ptr es:[10],11001111b  ; azzera quei bit
or              byte ptr es:[10],00100000b  ; bit per 16 colori 80x25
mov             ax,0003
int             10h                         ; seleziona modalità 3 a 16
                                            ;  colori 80x25
```

Funzione 1: impostazione dimensione cursore alfanumerico

Registri del chiamante:

AH = 1

CH = linea superiore del cursore

CL = linea inferiore del cursore

Valori restituiti:

(nessuno)

Aggiornamento dell'area dati di visualizzazione:

0040:0060CURSOR_MODE

La funzione 1 di INT 10H programma il controller del CRT per visualizzare il cursore alfanumerico specificato. Essa programma i registri di inizio cursore e di fine cursore del controller del CRT in modo che il cursore alfanumerico appaia tra le linee specificate nella matrice di carattere.

Il contenuto del registro CX viene copiato in CURSOR_MODE. Se il valore di CH è 20H, il cursore alfanumerico è disabilitato.

Su EGA e VGA, se il bit 0 del byte INFO (0040:0087) è impostato a 0, il BIOS elabora i valori di linea superiore e di linea inferiore passati in CH e CL relativi ad una matrice di carattere da otto linee. Il Capitolo 3 tratta dettagliatamente questa "emulazione di cursore".

Usate la funzione 1 di INT 10H solo nelle modalità video alfanumeriche. Per selezionare un cursore a piena altezza in modalità video 3 (modalità alfanumerica a 16 colori 80 per 25) su un CGA:

```
mov             cx,0007h            ; CH := 0 (linea superiore)
                                    ; CL := 7 (linea inferiore della matrice
                                    ;          di carattere 8x8)
mov             ah,1                ; AH := 1 (numero funzione INT 10H)
int             10h
```

Su EGA con un monitor a 350 linee, la modalità video 3 è una modalità alfanumerica a 350 linee con una matrice di carattere 8 per 14. Ciononostante, il codice sopra riportato opera normalmente senza modifiche in questa situazione in quanto il BIOS "emula" la corrispondente modalità a 200 linee del CGA e programma di conseguenza i registri di inizio cursore e di fine cursore.

Funzione 2: impostazione locazione cursore

Registri del chiamante:

AH = 2

BH = pagina video

DH = riga carattere

DL = colonna carattere

Valori restituiti:

(nessuno)

Aggiornamento area dati di visualizzazione:

0040:0050CURSOR_POSN

La funzione 2 di INT 10H aggiorna l'area dati di visualizzazione del BIOS, fornendo una nuova posizione del cursore. Se il valore di BH fa riferimento alla pagina correntemente visualizzata, questa routine programma anche il controller del CRT in modo che aggiorni la posizione del cursore visualizzato.

Per impostare la posizione del cursore alla colonna 10, riga 5, nella modalità a 16 colori 80 per 25:

```
mov         ah,2            ; AH := 2 (numero funzione INT 10H)
mov         bh,1            ; BH := pagina video
mov         dh,5            ; DH := riga
mov         dl,10           ; DL := colonna
int         10h
```

Funzione 3: ritorno dello stato del cursore

Registri del chiamante:

AH = 3

BH = numero di pagina video

Valori restituiti:

CH = linea superiore del cursore

CL = linea inferiore del cursore

DH = riga di carattere

DL = colonna di carattere

Aggiornamenti dell'area dati di visualizzazione:

(nessuno)

La funzione 3 di INT 10H restituisce la locazione del cursore di carattere per la pagina video specificata. I valori di riga e di colonna carattere vengono copiati da CURSOR_POSN nell'area dati di visualizzazione.

I valori ritornati in CH e CL vengono copiati da CURSOR_MODE, anch'esso nell'area dati di visualizzazione. Questi valori sono significativi solo nelle modalità alfanumeriche.

Per determinare la locazione corrente del cursore (e la dimensione in una modalità alfanumerica) nella pagina video 0:

```
mov             ah,3                ; AH := 3 (numero di funzione INT 10H)
mov             bh,0                ; BH := 0 (pagina video)
int             10h
```

Funzione 4: ritorno della posizione di penna ottica

Registri del chiamante:

AH = 4

Valori restituiti:

AH

= 1 se viene restituita una posizione valida di penna ottica
= 0 se non viene restituita una posizione di penna ottica

BX = coordinata *x* di pixe

CH = coordinata *y* di pixel (modalità video 4, 5 e 6 CGA e EGA)

CX = coordinata *y* di pixel (EGA tranne modalità video 4, 5 e 6)

DH = riga carattere

DL = colonna carattere

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 4 di INT 10H rileva la posizione corrente della penna ottica dai registri di penna ottica alta e di penna ottica bassa del controller del CRT.

Se il commutatore della penna ottica non è impostato, o se il latch di penna ottica non è stato attivato (cioè se i registri di penna ottica alto e di penna ottica basso del CRTC non contengono un indirizzo valido di penna ottica), la funzione 4 ritorna 0 nel registro AH. In caso contrario, la funzione 4 imposta a 1 AH, memorizza la posizione della penna ottica nei registri BX, CX e DX e ripristina il pulsante di attivazione della penna ottica.

Quando la funzione 4 termina, BX contiene la coordinata calcolata di pixel *x* alla quale la penna ottica è stata attivata. Dal momento che il CRTC ritorna la posizione di penna ottica sotto forma di indirizzo di byte, il valore di BX è preciso solo quanto il numero dei pixel in ogni byte del buffer del video (nella modalità a 2 colori 640 per 200

ogni byte del buffer del video rappresenta otto pixel; la funzione 4 ritorna quindi le coordinate *x* di ogni ottavo pixel). La posizione della penna ottica viene calcolata relativamente all'inizio della parte visualizzata del buffer del video (CRT_START).

La funzione 4 di INT 10H salva la coordinata *y* di pixel o in CH (nel BIOS della piastra madre) o in CX (in tutte le modalità video del BIOS dell'EGA ad eccezione delle modalità 4, 5 e 6). Ad esempio, nella modalità grafica a 4 colori 320 per 200, la coordinata *y* di pixel viene sempre salvata in CH, mentre nella modalità alfanumerica a 16 colori 80 per 25 il valore viene salvato in CH su un CGA ma in CX su un EGA.

I valori ritornati dalla funzione 4 in DH e DL rappresentano la riga e la colonna di carattere alle quali la penna ottica è stata attivata. La funzione 4 di INT 10H ritorna sempre AH = 0 su MCGA e VGA, che non supportano la penna ottica.

Per determinare lo stato della penna ottica in qualsiasi modalità video, richiamate la funzione 4 di INT 10H:

```
mov             ah,4                ; AH := 4 (numero funzione INT 10H)
int             10h
```

Ad esempio, se attivate la penna ottica vicino al centro dello schermo nella modalità a 16 colori 640 per 350, i valori ritornati da questa funzione potrebbero essere:

AH = 1 (risultati validi di penna ottica ritornati)

BX = 320 (coordinata *x* del primo pixel all'indirizzo di dove la penna è stata attivata)

CX = 175 (coordinata *y* di pixel)

DH = 12 (riga carattere)

DL = 40 (colonna carattere)

Funzione 5: selezione pagina video

Registri del chiamante:

AH = 5

AL = numero pagina video

Valori restituiti:

(nessuno)

Aggiornamenti area dati di visualizzazione:

0040:004ECRT_START
0040:0062ACTIVE_PAGE

La funzione 5 di INT 10H seleziona quale parte del buffer del video viene visualizzata su CGA, EGA, MCGA e VGA. Essa opera programmando i registri di inizio indirizzo del CRTC. Potete usare questa funzione nelle modalità alfanumeriche 40 per 25 o 80 per 25 (modalità BIOS 0, 1, 2 e 3) in tutti questi sistemi.

Su CGA, l'intero buffer del video da 16 KB viene utilizzato nelle modalità grafiche 320 per 200 e 640 per 200, quindi non è possibile avere più pagine video. Le chiamate alla funzione 5 vengono ignorate in queste modalità.

Su MCGA, EGA e VGA, le pagine video sono disponibili sia nelle modalità alfanumeriche sia in quelle grafiche fino ai limiti della RAM del video. Tuttavia, la routine del BIOS non controlla se la RAM del video è sufficiente per supportare una pagina video richiesta; se la pagina video richiesta si trova al di fuori del buffer del video, la visualizzazione risultante non avrà significato.

Il BIOS mantiene una locazione di cursore corrente per un massimo di otto pagine video in CURSOR_POSN. Quando richiamate la funzione 5, il BIOS sposta il cursore sulla posizione alla quale si trovava l'ultima volta che è stata visualizzata la pagina richiesta.

La seguente routine imposta la parte visualizzata del buffer del video della CGA in modo che inizi a B800:1000 (pagina video 1) nella modalità alfanumerica 80 per 25:

```
mov             ax,0501h            ; AH := 5 (numero di funzione INT 10H)
                                    ; AL := 1 (numero pagina video)
int             10h
```

Funzione 6: Scorrimento verso l'alto

Registri del chiamante:

AH = 6

AL = numero di linee da far scorrere

BH = attributo

CH = riga dell'angolo superiore sinistro

CL = colonna dell'angolo superiore sinistro

DH = riga dell'angolo inferiore destro

DL = colonna dell'angolo inferiore destro

Valori restituiti:

(nessuno)

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 6 di INT 10H esegue uno scorrimento verso l'alto riga per riga dei caratteri nell'area indicata della pagina video attiva. Voi specificate il numero di righe di caratteri da far scorrere in AL. L'area rettangolare nella quale deve avvenire lo scorrimento viene definita dal suo angolo superiore sinistro, specificato in CH e CL, e dal suo angolo inferiore destro, specificato in DH e DL.

L'attributo specificato in BH viene usato per tutte le linee vuote inserite nella parte inferiore dell'area che viene fatta scorrere. Nelle modalità alfanumeriche, questo attributo viene formattato nella solita maniera, con l'attributo di sfondo nel semibyte più significativo e l'attributo di primo piano nel semibyte meno significativo. Nelle modalità grafiche, il formato dell'attributo in BH dipende dalla modalità.

Nelle modalità a 2 colori 640 per 200 e a 4 colori 320 per 200, il valore di BH rappresenta una configurazione di pixel a 1 byte. Il byte rappresenta otto pixel da 1 bit nella modalità a 2 colori 640 per 200 o quattro pixel a 2 bit nella modalità a 4 colori 320 per 200. La configurazione di pixel viene duplicata per tutte le linee che la funzione 6 cancella nell'area di scorrimento. In tutte le altre modalità grafiche EGA, MCGA e VGA il valore di BH determina il valore di tutti i pixel nelle linee cancellate.

Nella modalità a 4 colori 320 per 200 su EGA, MCGA e VGA, la funzione 6 produce sempre lo scorrimento della pagina video 0, a prescindere da quale pagina video sia visualizzata correntemente.

La specifica di 0 in AL come numero di righe da far scorrere produce la cancellazione dell'intera area di scorrimento.

Nella modalità alfanumerica a 16 colori 80 per 25 potete far scorrere l'intero schermo verso l'alto di una riga con la sequenza:

```
mov     ax,601h     ; AH := 6 (numero funzione INT 10H)
                    ; AL := 1 (numero di linee da far
                    ;          scorrere verso l'alto)
mov     bh,7        ; BH := 7 (attributo)
mov     cx,0        ; CH := angolo superiore sinistro:
                    ;       riga 0
                    ; CL := angolo superiore sinistro:
                    ;       colonna 0
mov     dx,184Fh    ; DH := angolo inferiore destro:
                    ;       riga 24 (18H)
                    ; DL := angolo inferiore destro:
                    ;       colonna 79 (4FH)
int     10h
```

Nella stessa modalità video, potete cancellare solo le tre righe superiori dello schermo con un attributo di sfondo 1 (blu su CGA) e un attributo di primo piano 7 (bianco) usando la seguente routine.

```
mov             ax,600h             ; AH := numero funzione INT 10H
                                    ; AL := 0 (cancella l'area di scorrimento)
mov             bh,17h              ; BH := attributo (sfondo 1, primo piano 7)
mov             cx,0                ; CH,CL := angolo superiore sinistro a (0,0)
mov             dx,024Fh            ; DH,DL := angolo inferiore destro a (2,79)
int             10h
```

Per ottenere lo stesso risultato nella modalità grafica a 16 colori 640 per 350 su EGA, impostate il valore di BH per indicare un valore di pixel invece di un attributo alfanumerico:

```
mov             ax,600h
mov             bh,1                ; BH := valore di pixel
mov             cx,0
mov             dx,024Fh
int             10h
```

Nella modalità a 2 colori 640 per 200, la seguente chiamata alla funzione 6 di INT 10H riempie lo schermo con strisce verticali di valori di pixel alternati:

```
mov             ax,600h
mov             bh,10101010b        ; BH := configurazione di pixel
mov             cx,0
mov             dx,184Fh
int             10h
```

Funzione 7: scorrimento verso il basso

Registri del chiamante:

AH = 7

AL = numero di linee da far scorrere

BH = attributo

CH = riga dell'angolo superiore sinistro

CL = colonna dell'angolo superiore sinistro

DH = riga dell'angolo inferiore destro

DL = colonna dell'angolo inferiore destro

Valori restituiti:

(nessuno)

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 7 di INT 10H esegue uno scorrimento riga per riga verso il basso dei caratteri di un'area indicata della pagina video attiva. Ad eccezione della direzione dello scorrimento, questa funzione del BIOS è identica alla funzione 6.

Funzione 8: ritorno di codice e di attributo di carattere alla posizione del cursore

Registri del chiamante:

AH = 8

BH = pagina video

Valori restituiti:

AH = attributo (solo modalità alfanumeriche)

AL = codice ASCII

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 8 di INT 10H ritorna il codice ASCII del carattere alla posizione corrente del cursore nella pagina video specificata da BH. Nelle modalità alfanumeriche, ciò viene realizzato leggendo una singola word dal buffer del video. Nelle modalità grafiche, la routine confronta la matrice di carattere alla posizione del cursore con le configurazioni di bit della corrente tabella di definizione caratteri grafici.

Nelle modalità grafiche, il BIOS del PC/XT e del PC/AT usa le definizioni di carattere ROM a F000:FA6E; il BIOS dell'EGA, dell'MCGA e della VGA usa definizioni indicate dal vettore di interrupt 43H. Per i codici ASCII da 80 a 0FFH nelle modalità grafiche 4, 5 e 6 compatibili CGA, il BIOS usa i caratteri definiti nella tabella indicata dal vettore di interrupt 1FH.

Per determinare il codice di carattere per un carattere in una modalità grafica, la routine del BIOS considera i pixel non zero come pixel di primo piano. Vengono confrontati i pixel della configurazione di primo piano (non zero) e di sfondo (zero) con le configurazioni di bit della tabella. Se la configurazione di pixel del buffer del video corrisponde ad una configurazione di bit nella tabella di definizione caratteri, il BIOS determina il codice ASCII del carattere dalla locazione della configurazione di bit nella tabella. Se la configurazione di pixel nel buffer del video non corrisponde a nessuna configurazione di bit della tabella, la routine del BIOS ritorna 0 in AL.

Nella modalità a 4 colori 320 per 200 su EGA, MCGA e VGA, questa funzione opera correttamente solo nella pagina video 0.

Il seguente frammento di codice legge il carattere nell'angolo superiore sinistro dello schermo:

```
mov         ah,0Fh          ; AH := 0FH (numero funzione INT 10H)
int         10h             ; lascia BH = pagina video attiva
mov         ah,2            ; AH := 2 (numero funzione INT 10H)
mov         dx,0            ; DH,DL := riga 0, colonna 0
int         10h             ; imposta posizione cursore a (0,0)
mov         ah,8            ; AH := 8 (numero funzione INT 10H)
int         10h             ; lascia AL = codice ASCII
```

Funzione 9: scrittura carattere ed attributo alla posizione del cursore

Registri del chiamante:

AH = 9

AL = codice ASCII

BH = valore di pixel di sfondo (modalità a 256 colori 320 per 200) o pagina video (tutte le altre modalità)
BL = valore di pixel di primo piano (modalità grafiche) o valore di attributo (modalità alfanumeriche)
CX = fattore di ripetizione

Valori restituiti:

(nessuno)

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 9 di INT 10H scrive un carattere una o più volte nel buffer del video senza spostare il cursore. Dovete specificare un fattore di ripetizione uguale o superiore a 1 in CX. Il BIOS scrive nel buffer una stringa composta dal carattere in AL. La lunghezza della stringa viene determinata dal fattore di ripetizione in CX.

Nelle modalità alfanumeriche, sia il codice ASCII sia il byte di attributo corrispondente vengono aggiornati per ogni carattere scritto nel buffer del video. Nelle modalità grafiche ogni carattere viene scritto nel buffer in un'area rettangolare con una dimensione pari a quella della matrice di carattere. Il valore di BL viene utilizzato per i pixel di primo piano del carattere. Nella modalità grafica a 256 colori 320 per 200, il valore di BH specifica il valore del pixel di sfondo; in tutte le altre modalità grafiche, BH indica una pagina video , quindi i pixel di sfondo del carattere sono 0. In tutte le modalità grafiche ad eccezione della modalità a 256 colori 320 per 200, il carattere viene sottoposto a XOR nel buffer se il bit 7 di BL è impostato a 1.

La funzione 9 di INT 10H non confronta il fattore di ripetizione con il numero di colonne di caratteri visualizzati. Nelle modalità alfanumeriche, questo potrebbe non essere rilevante; la mappa del buffer del video è strutturata in modo tale che una stringa troppo lunga per essere visualizzata in una riga di caratteri prosegue sulla riga successiva. Nelle modalità grafiche, però, una stringa non dovrebbe essere più lunga del resto della riga corrente di caratteri.

Dovete specificare una pagina video nel registro BH nelle modalità alfanumeriche oltre che nelle modalità grafiche EGA originali, ma il valore di BH viene ignorato dal BIOS dell'EGA, dell'MCGA e della VGA nella modalità grafica a 4 colori 320 per 200.

La seguente routine scrive una stringa di 20 asterischi nell'angolo superiore sinistro dello schermo nella modalità a 16 colori 80 per 25. Il valore di primo piano del byte di attributo di ogni carattere è impostato a 7 e il valore di sfondo è impostato a 1. Il cursore viene posizionato con una chiamata alla funzione 2 di INT 10H prima che la stringa venga scritta con la funzione 9.

```
mov         ah,2            ; AH := 2 (numero funzione INT 10H)
mov         bh,0            ; BH := pagina video
mov         dx,0            ; DH := riga cursore
                            ; DL := colonna cursore
int         10h             ; imposta posizione cursore a (0,0)
mov         ah,9            ; AH := 9 (numero funzione INT 10H)
mov         al,'*'          ; AL := codice ASCII
mov         bl,17h          ; BL := byte di attributo
mov         cx,20           ; CX := fattore di ripetizione
int         10h
```

Funzione 0AH: scrittura di carattere (o caratteri) alla posizione del cursore

Registri del chiamante:

AH = 0AH

AL = codice ASCII

BH = valore di pixel di sfondo (modalità a 256 colori 320 per 200) o pagina video (tutte le altre modalità)

BL = valore di pixel di primo piano (solo modalità grafiche)

CX = fattore di ripetizione

Valori restituiti:

(nessuno)

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 0AH di INT 10H è identica alla funzione 9 di INT 10H, con una eccezione: nelle modalità video alfanumeriche, solo il codice di carattere viene scritto nel buffer del video. L'attributo del carattere resta immutato nel buffer.

Questo esempio cancella una riga di caratteri a partire dalla posizione del cursore fino alla fine della riga. Prima di chiamare la funzione 0AH, l'esempio determina la pagina video attiva ed il numero di colonne di caratteri visualizzati con una chiamata alla funzione 0FH di INT 10H, e la posizione del cursore usando la funzione 3 di INT 10H.

```
mov         ah,0Fh          ; AH := 0FH (numero di funzione INT 10H)
int         10h             ; lascia AH = numero di colonne
                            ; BH = pagina video attiva
mov         al,ah
xor         ah,ah           ; AX := numero di colonne
push        ax
mov         ah,3            ; AH := 3 (numero funzione INT 10H)
int         10h             ; lascia DH,DL = posizione del cursore
pop         cx              ; CX := colonne di caratteri visualizzati
sub         cl,dl           ; CX := numero di caratteri restanti
                            ; nella riga
xor         bl,bl           ; BL := valore di pixel di primo piano
mov         ax,0A20h        ; AH := 0AH (numero funzione INT 10H)
                            ; AL := 20H (carattere spazio ASCII)
int         10h
```

Funzione 0BH: impostazione colore sovrascansione, selezione tavolozza a 4 colori

Registri del chiamante:

AH = 0BH

BH = 0 per impostare colore di cornice o di sfondo
= 1 per selezionare tavolozza a 4 colori

BL = valore colore (se BH=0)
= valore tavolozza (se BH=1)

Valori restituiti:

(nessuno)

Aggiornamento area dati di visualizzazione:

0040:0066 CRT_PALETTE

La funzione 0BH di INT 10H comprende due sottofunzioni selezionate secondo il valore di BH. La funzione 0BH è stata progettata per essere usata solo nella modalità a 4 colori 320 per 200 e nelle modalità alfanumeriche CGA, ma potete usarla anche, con precauzione, nelle altre modalità video.

BH=0

Quando BH=0 su CGA e MCGA, il BIOS carica i cinque bit meno significativi del valroe di BL nel registro di selezione colore (3D9H). Nella modalità grafica a 4 colori 320 per 200, i bit da 0 a 3 determinano il colore di sfondo (il colore visualizzato per i pixel con valore 0) oltre che il colore di cornice. Nelle modalità a 2 colori 640 per 200 e 640 per 480, i bit da 0 a 3 specificano il colore dei pixel di primo piano (non zero). Su CGA, questi stessi quattro bit determinano anche il colore di cornice nelle modalità alfanumeriche.

Il bit 4 del registro di selezione colore indica l'uso di colori normali oppure evidenziati nelle modalità grafiche CGA e MCGA (vedere Capitolo 4). Per mantenere la compatibilità, il BIOS dell'EGA e della VGA emula questo effetto usando una tavolozza di colori evidenziati quando il bit 4 di BL è impostato.

Nelle modalità a 200 linee su EGA e VGA, il valore di BL viene posto nel registro di colore sovrascansione (11H) del controller di attributo. Questo imposta il colore di cornice. Se uno dei sistemi di visualizzazione si trova in una modalità grafica, lo stesso valore viene memorizzato anche nel registro tavolozza 0. Ciò stabilisce lo stesso colore per tutti i pixel di valore 0.

Non utilizzate la funzione 0BH con BL=0 in altre modalità video EGA e VGA. In alcune modalità, la routine del BIOS memorizza valori di colore errati nei registri tavolozza e di sovrascansione, mentre in altre non effettua alcuna operazione. Dovreste utilizzare la funzione 10H di INT 10H per programmare il controller di attributo su EGA e VGA.

Una volta programmato il registro di colore o il controller di attributo, la routine del BIOS copia il bit 5 di CRT_PALETTE nell'area dati di visualizzazione sul bit 0 del registro BL e trasferisce il controllo alla routine per BH=1.

BH=1

Quando BH=1, il bit meno significativo del valore di BL determina quale delle due tavolozze a 4 colori viene utilizzata per la modalità a 4 colori 320 per 200 (vedere Figura 144). Su CGA e MCGA questo bit viene copiato nel bit 5 del registro di selezione colore (3D9H). Su EGA e VGA, il bit determina quale set di valori di colore viene caricato nei registri di tavolozza del controller di attributo. I colori corrispondono alle tavolozze a 4 colori 320 per 200 del CGA (vedere Capitolo 4 per ulteriori dettagli).

Valore di pixel (bit 0 di BL = 0)	Colore visualizzato
1	Verde
2	Rosso
3	Giallo
Valore di pixel (bit 0 di BL = 1)	**Colore visualizzato**
1	Ciano
2	Viola
3	Bianco

Figura 144. *Tavolozze a 4 colori della funzione 0BH.*

La funzione 0BH con BH=1 non ha alcun effetto nelle modalità alfanumeriche. Nelle modalità grafiche diverse dalla modalità a 4 colori 320 per 200, comunque, viene caricato il registro di selezione colore (su CGA e MCGA) o vengono aggiornati i registri di tavolozza (su EGA e VGA) come se fosse attiva la modalità a 4 colori 320 per 200. Per questo motivo, dovreste utilizzare questa sottofunzione con cautela nelle modalità grafiche diverse dalla modalità a 4 colori 320 per 200.

L'esempio che segue ha tre diversi effetti, a seconda della modalità video corrente. Nelle modalità alfanumeriche a 200 linee, imposta il colore di cornice; nella modalità a 4 colori 320 per 200 imposta sia il colore di cornice sia il colore di sfondo; infine nelle modalità grafiche a 2 colori CGA o MCGA, imposta il colore di primo piano.

```
mov             ah,0BH              ; AH := 0BH (numero funzione INT 10H)
mov             bh,0                ; BH := numero sottofunzione
mov             bl,BorderColor      ; BL := valore colore
int             10h
```

Per selezionare una tavolozza a 4 colori nella modalità a 4 colori 320 per 200, richiamate la funzione 0BH con BH=1:

```
mov             ah,0Bh
mov             bh,1                ; BH := numero sottofunzione
mov             bl,0                ; bit 0 di BL := 0 (tavolozza
                                    ;                   rosso-verde-giallo)
int             10h
```

Nella modalità a 4 colori 320 per 200, selezionate un set di colori evidenziati richiamando la funzione 0BH con BH=0 e con il bit 4 di BL impostato a 1.

```
mov             ah,0Bh
mov             bh,0
mov             bl,10h              ; il bit 4 seleziona tavolozza evidenziata
                                    ; i bit 3-0 selezionano colore
                                    ; cornice/sfondo
int             10h
```

Funzione 0CH: memorizzazione valore di pixel

Registri del chiamante:

AH = 0CH

AL = valore di pixel

BH = pagina video

CX = coordinata *x*

DX = coordinata *y*

Valori restituiti:

(nessuno)

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 0CH di INT 10H aggiorna il valore di un pixel ad una locazione specificata nel buffer del video. In tutte le modalità grafiche ad eccezione della modalità a 256 colori 320 per 200, se il bit più significativo del valore di AL è impostato a 1, il valore di AL viene sottoposto a XOR nel buffer video. In caso contrario, il valore di AL diventa il nuovo valore di pixel.

Su EGA, MCGA e VGA il valore di BH viene utilizzato per selezionare tra le pagine video disponibili nella modalità video corrente. Tuttavia, il valore di BH viene ignorato nella modalità a 4 colori 320 per 200.

Per impostare il valore di un pixel in una modalità grafica a 350 linee su EGA con solo 64 KB di RAM video, dovete tener conto dei concatenamenti delle mappe di memoria con i piani di bit (come visto nel Capitolo 4). In questa situazione, la routine del BIOS si aspetta che voi specifichiate il valore di pixel in AL utilizzando solo i suoi bit con numeri dispari. Di conseguenza, i quattro possibili valori di pixel dovrebbero essere specificati come 0 (0000B), 1 (0001B), 4 (0100B) e 5 (0101B) al posto di 0, 1, 2 e 3.

La seguente routine mostra come impostare a 1 un valore di pixel a (200,100) in qualsiasi modalità grafica:

```
mov             ah,0Ch              ; AH := 0CH (numero funzione INT 10H)
mov             al,1                ; AL := valore di pixel
mov             cx,200              ; CX := coordinata x
mov             dx,100              ; DX := coordinata y
int             10h
```

Per effettuare l'XOR del valore di un pixel nel buffer del video, impostate a 1 il bit 7 di AL prima di eseguire l'interrupt 10H, come nella seguente procedura:

```
mov             ah,0Ch
mov             al,1
mov             cx,200
mov             dx,100
or              al,10000000b        ; imposta il bit 7 per indicare XOR
int             10h
```

Questo frammento di codice illustra la speciale situazione che sorge in una modalità video a 350 linee su EGA IBM con solo 64 KB di RAM video. Il codice imposta a 3 il valore del pixel a (75,50) .

```
mov             ah,0Ch
mov             al,0101b                ; AL := valore di pixel di 3 (11B)
                                        ; rappresentato solo da bit dispari
mov             cx,75
mov             dx,50
int             10h
```

Funzione 0DH: restituzione del valore di un pixel

Registri del chiamante:

AH = 0DH

BH = pagina video

CX = coordinata *x*

DX = coordinata *y*

Valori restituiti:

AL = valore di pixel

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 0DH di INT 10H ritorna il valore di un pixel ad una locazione specificata nel buffer del video.

Su EGA in modalità a 4 colori 320 per 200 la funzione ignora il valore di pagina video specificato in BH.

Il BIOS EGA IBM (versione 13/9/84) contiene un errore nella funzione 0DH di INT 10H. Nelle modalità grafiche a 350 linee su EGA IBM con solo 64 KB di RAM video, il valore ritornato in AL è errato. Apparentemente, la routine del BIOS calcola l'offset di

byte del pixel nel buffer del video senza tener conto in modo appropriato della mappatura degli indirizzi pari con i piani di bit pari e degli indirizzi dispari con i piani di bit dispari.

Per determinare il valore del pixel a (100,100) potreste eseguire la sequenza di istruzioni riportata qui sotto:

```
mov         ah,0Dh          ; AH := 0DH (numero funzione INT 10H)
mov         bh,0            ; BH := pagina video (0 in questo esempio)
mov         cx,100          ; CX := coordinata x
mov         dx,100          ; DX := coordinata y
int         10h             ; lascia AL = valore di pixel
```

Funzione 0EH: visualizzazione caratteri in modalità teletype

Registri del chiamante:

AH = 0EH

AL = codice ASCII

BH = pagina video (BIOS PC versioni datate 19/10/81 o precedenti)

BL = valore di pixel di primo piano (solo modalità grafiche)

Valori restituiti:

(nessuno)

Aggiornamento area dati di visualizzazione:

0040:0050 CURSOR_POSN

La funzione 0EH di INT 10H richiama la funzione 0AH di INT 10H per visualizzare il carattere da passare nel registro AL. A differenza della funzione 0AH, però, la funzione 0EH sposta il cursore, e tratta i codici ASCII 7 (avviso acustico), 8 (backspace), 0DH (ritorno carrello) e 0AH (avanzamento riga) come comandi di controllo cursore invece che come caratteri visualizzabili. La funzione 0EH aggiorna sempre la pagina video attiva (visualizzata correntemente) tranne che nei casi citati prima.

Se il carattere viene visualizzato nella colonna di caratteri all'estrema destra, la funzione 0EH sposta in avanti il cursore fino all'inizio della riga di caratteri successiva. Se necessario, la funzione 0EH richiama la funzione 06H di INT 10H per effettuare lo scorrimento dello schermo. Nelle modalità alfanumeriche per lo scorrimento viene usato l'attributo del carattere visualizzato. Nelle modalità grafiche, l'attributo di scorrimento è sempre 0.

Nelle modalità alfanumeriche, il byte di attributo alla posizione in cui viene scritto il carattere determina gli attributi di primo piano e di sfondo del carattere. Per questo motivo, dovreste probabilmente riempire il buffer del video con gli attributi alfanumerici desiderati prima di utilizzare la funzione 0EH.

Nelle modalità grafiche, il carattere viene scritto nel buffer del video in un'area rettangolare che ha la dimensione della matrice di carattere. I pixel del carattere hanno il valore specificato da BL, e i restanti pixel di sfondo hanno il valore di 0. Dal momento che il valore di BL viene passato alla funzione 0AH di INT 10H, potete impostare il bit 7 in modo che il carattere venga sottoposto a XOR nel buffer del video.

NOTA: Sfortunatamente, la funzione 0EH non espande i caratteri di tabulazione (codice ASCII 9) in spazi.

La seguente routine mostra come potreste usare la funzione 0EH per visualizzare una stringa di caratteri.

```
            mov     cx,StringLength       ; CX := numero di byte
                                          ;       nella stringa
            jcxz    L02                   ; non esegue nulla se stringa
                                          ;  vuota
            mov     si,StringAddr         ; DS:SI := indirizzo della stringa
            mov     bl,GraphicsAttribute  ; BL := attributo (solo
                                          ;       modalità grafiche)
L01:        lodsb                         ; AL := carattere successivo
                                          ;       nella stringa
            mov     ah,0Eh                ; AH := 0EH (numero funzione
                                          ;            INT 10H)
            int     10h
            loop    L01
L02:        .
            .
            .
```

Funzione 0FH: ritorno dello stato corrente del video

Registri del chiamante:

AH = 0FH

Valori restituiti:

AH = numero di colonne di caratteri visualizzati

AL = numero modalità video

BH = pagina video attiva

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 0FH di INT 10H fornisce informazioni sulla modalità video corrente e sulla larghezza della parte visualizzata del buffer del video. Il numero di colonne di caratteri (ritornato in AH) e il numero di pagina video corrente (in BH) vengono copiati da CRT_COLS e da ACTIVE_PAGE nell'area dati di visualizzazione.

Il valore restituito in AL viene copiato da CRT_MODE nell'area dati di visualizzazione. Esso corrisponde alle modalità video tabulate per la funzione 0. Su EGA e VGA, il bit 7 del valore in AL viene ricavato dal bit 7 del byte INFO (la funzione 0 di INT 10H imposta il bit 7 del byte INFO ogni volta che usate la funzione 0 per selezionare una modalità video senza cancellare il buffer del video).

Questo esempio mostra come determinare la posizione corrente del cursore visualizzato. Prima di richiamare la funzione 3 di INT 10H per trovare la posizione del cursore, l'esempio usa la funzione 0FH per determinare la pagina visualizzata correntemente.

```
mov         ah,0Fh          ; AH := 0FH (numero funzione INT 10H)
int         10h             ; lascia BH = pagina video attiva
mov         ah,3            ; AH := 3 (numero funzione INT 10H)
int         10h             ; lascia DH,DL = posizione cursore
```

Funzione 10H: impostazione registri di tavolozza, impostazione attributo di evidenziazione/lampeggiamento

Registri del chiamante:

AH = 10H

Aggiorna un registro di tavolozza specificato:

AL = 0

BH = valore colore

BL = numero registro tavolozza

Specifica colore di sovrascansione (cornice):

AL = 1

BH = valore colore

Aggiorna tutti i 16 registri di tavolozza oltre al registro di sovrascansione:

AL = 2

ES:DX = indirizzo di tabella a 17 byte

Seleziona evidenziazione sfondo o attributo lampeggiamento:

AL = 3

BL
= 0 per evidenziazione sfondo (lampeggiamento disabilitato)
= 1 per lampeggiamento

Legge un registro di tavolozza specificato:

AL = 7

BL = numero registro di tavolozza

Valore restituito:

BH = contenuto del registro di tavolozza specificato

Legge il contenuto del registro di sovrascansione:

AL = 8

Valore restituito:

BH = contenuto del registro di sovrascansione

Legge tutti i 16 registri di tavolozza oltre al registro di sovrascansione:

AL = 9

ES:DX = indirizzo della tabella a 17 byte

Valori restituiti:

I byte da 00H a 0FH della tabella contengono i valori di registro tavolozza

Il byte 10H della tabella contiene il valore del registro di sovrascansione.

Aggiorna il registro di colore del DAC del video specificato:

AL = 10H

BX = numero registro colore

CH = valore verde

CL = valore blu

DH = valore rosso

Aggiorna un blocco di registri di colore del DAC del video:

AL = 12H

BX = primo registro da aggiornare

CX = numero di registri da aggiornare

ES:DX = indirizzo della tabella dei valori rosso-verde-blu

Imposta stato selezione colore del controller di attributo:

AL = 13H

BL = 0 per impostare il bit 7 del registro di controllo modalità, 1 per impostare il registro di selezione colore

BH = valore per il bit 7 (se BL=0) o valore per registro di selezione colore (se BL=1)

Legge registro di colore del DAC del video specificato:

AL = 15H

BX = numero registro colore

Valori restituiti:

CH = verde

CL = blu

DH = rosso

Legge un blocco di registri di colore del DAC del video:

AL = 17H

BX = primo registro da leggere

CX = numero di registri da leggere

ES:DX = indirizzo della tabella dei valori rosso-verde-blu

Valori restituiti:

I byte da 0 a $3n$-1 (dove n è il numero di registri passati a CX) contengono i valori di rosso-verde-blu letti dal blocco specificato dei registri di colore.

Aggiorna il registro di maschera del DAC del video:

AL = 18H

BL = nuovo valore di maschera

Legge registro di maschera del DAC del video:

AL = 19H

Valori restituiti:

BL = valore letto dal registro di maschera del DAC del video

Legge registro di selezione colore del controller di attributo:

AL = 1AH

Valori restituiti:

BL = bit 7 del registro di controllo modalità

BH = bit 2 e 3 del registro di selezione colore (se BL=0)
bit da 0 a 3 del registro di selezione colore (se BL=1)

Esegue calcolo trasformazione in scala di grigi su un blocco di registri di colore del DAC del video:

AL = 1BH

BX = primo registro di colore del blocco

CX = numero dei registri di colore

Aggiornamenti area dati di visualizzazione:

0040:0065CRT_MODE_SET
0040:0066CRT_PALETTE

La funzione 10H di INT 10H esiste solo nel BIOS dell'EGA, dell'MCGA e della VGA. La funzione comprende 16 sottofunzioni che vengono selezionate a seconda del valore contenuto in AL. La Figura 145 mostra il supporto che i vari sistemi forniscono a queste sottofunzioni. Tutte le sottofunzioni operano sia nelle modalità grafiche sia in quelle alfanumeriche.

Le sottofunzioni da 0 a 9 supportano la programmazione dell'attributo e della tavolozza. Le sottofunzioni da 10H a 1BH supportano il DAC del video su MCGA e VGA.

AL=0

Quando AL=0 su EGA e VGA, la funzione 10H aggiorna il valore di uno dei registri di tavolozza nel controller di attributo. La routine carica il valore di BH nel registro specificato da BL.

Sebbene lo scopo di questa sottofunzione sia quello di caricare un valore di colore in un registro di tavolozza, la routine del BIOS non convalida il numero di registro di BL. Di conseguenza, potete anche utilizzare questa sottofunzione per aggiornare i registri di controllo modalità, di sovrascansione, di abilitazione piano colore e di scorrimento pixel orizzontale del controller di attributo.

Su MCGA, quando BH=7 e BL=12H, la routine del BIOS imposta a 0 il bit 3 del registro di maschera del DAC del video (3C6H). Questo fa sì che il BIOS consideri il bit 3 di tutti i valori di pixel o degli attributi alfanumerici a 4 bit come bit "ininfluente" in riferimento ai registri di colore del DAC del video, quindi è possibile far riferimento solo ai primi otto registri. Ciò è utile nella visualizzazione di due set di 256 caratteri in una modalità alfanumerica (vedere Capitolo 10). Il BIOS dell'MCGA ignora tutti gli altri valori di BH o BL.

Sottofunzione	EGA	MCGA	VGA
0	x	x	x
1	x		x
2	x		x
3	x	x	x
4 (riservato)			
5 (riservato)			
6 (riservato)			
7			x
8			x
9			x
10H		x	x
11H (riservato)			
12H		x	x
13H			x
14H (riservato)			
15H		x	x
16H (riservato)			
17H		x	x
18H		x	x
19H		x	x
1AH			x
1BH		x	x

Figura 145. *Supporto della funzione 10H di INT 10H nel BIOS dell'EGA, dell'MCGA e della VGA.*

AL=1

Quando AL=1 su EGA e VGA, il BIOS copia il valore di BH nel registro di sovrascansione (11H) del controller di attributo.

AL=2

Quando AL=2 su EGA e VGA, il BIOS presume che ES:DX contenga l'indirizzo di una tabella a 17 byte di valori per i 16 registri di tavolozza (i byte da 0 a 15) e per il registro di sovrascansione (il byte 16). La routine copia questi valori nei registri corrispondenti nel controller di attributo.

AL=3

Quando AL=3 su EGA e VGA, il valore di BL determina il valore del bit 3 del registro di controllo modalità (10H) del controller di attributo. Se BL=0, il bit 3 del valore del registro di controllo modalità viene impostato a 0, disabilitando l'attributo di lampeggiamento. Se BL è 1, il bit 3 viene impostato a 1 per abilitare il lampeggiamento.

Quando AL=3 su MCGA, il bit 5 del registro di controllo colore (3D8H) viene impostato in modo che rispecchi il valore di BL. Se BL=0, il bit 5 viene impostato a 0 per disabilitare il lampeggiamento. Se BL è 1, il bit 5 viene impostato a 1.

AL=7

Quando AL=7 su VGA, il valore del registro di tavolozza del controller di attributo specificato da BL viene ritornato in BH. Dal momento che il BIOS non verifica il numero di registro specificato, questa sottofunzione può essere utilizzata per verificare il contenuto di qualsiasi registro del controller di attributo.

AL=8

Quando AL=8 su VGA, il contenuto del registro di sovrasacansione del controller di attributo viene restituito in BH.

AL=9

Quando AL=9 su VGA, il contenuto di tutti i 16 registri di tavolozza e del registro di sovrascansione viene restituito ad una tabella a 17 byte il cui indirizzo è stato indicato al BIOS nella coppia di registri ES:DX.

AL=10H

Quando AL=10H su MCGA e VGA, il registro di colore del DAC del video specificato da BX viene aggiornato con i valori di rosso, verde e blu specificati in DH, CH e CL. Vengono considerati solo i sei bit meno significativi di ognuno dei tre valori di colore.

Se è abilitato il calcolo della scala di grigi, il valore memorizzato nel registro di colore è il valore di scala di grigi corrispondente ai valori di colore specificati (vedere funzione 12H di INT 10H con BL=33H).

AL=12H

Quando AL=12H su MCGA e VGA, un blocco di registri consecutivi di colore del DAC del video viene aggiornato dalla tabella il cui indirizzo viene indicato in ES:DX. Il valore di BX (da 00H a 0FFH) indica il primo registro di colore da aggiornare, e CX contiene il numero di registri interessati. La routine del BIOS non esegue alcun controllo errori; se la somma dei valori di BX e di CX è maggiore di 256 (100H), la routine ricomincia da capo e aggiorna il primo registro (o i primi registri) di colore del DAC del video.

Se viene abilitato il calcolo della scala di grigi, i valori memorizzati nei registri di colore sono i valori di scala di grigi corrispondenti ai valori di colore della tabella (vedere funzione 12H di INT 10H con BL=33H).

Dovete formattare la tabella in gruppi di tre byte. Ogni gruppo deve contenere un valore di rosso nel primo byte, un valore di verde nel secondo ed un valore di blu nel terzo. Vengono considerati solo i sei bit meno significativi di ogni valore di colore.

AL=13H

Su VGA, quando AL=13H, il BIOS ROM aggiorna il registro di controllo modalità (10H) del controller di attributo ed il registro di selezione colore (14H) per abilitare il raggruppamento dei 256 registri di colore del DAC del video in blocchi di 16 o di 64 registri ognuno, come visto nel Capitolo 3.

Se BL=0, il BIOS usa il valore indicato in BH per aggiornare il bit 7 del registro di controllo modalità, se BH=1, il bit 7 viene impostato a 1. Questo fa sì che il BIOS usi i bit 0 e 1 del registro di selezione colore al posto dei bit 4 e 5 dei valori di registro tavolozza. Quando BH=0, il bit 7 viene impostato a 0 e vengono presi in considerazione tutti i sei bit meno significativi dei valori nei registri di tavolozza.

Quando BL=1, il valore di BH viene memorizzato negli appropriati campi di bit del registro di selezione colore. Se il bit 7 del registro di controllo modalità è 1, i bit da 0 a 3 del valore di BH vengono copiati nei bit da 0 a 3 del registro di selzione colore. Se il bit 7 del registro di controllo modalità è 0, i bit 0 e 1 di BH vengono copiati nei bit 2 e 3 del registro di selezione colore.

AL=15H

Quando AL=15H su MCGA e VGA, il contenuto del registro di colore del DAC del video specificato in BX viene ritornato nei registri DH (rosso), CH (verde) e CL (blu). Vengono considerati solo i bit meno significativi di ogni valore di colore.

AL=17H

Quando AL=17H su MCGA e VGA, i valori di un blocco di registri adiacenti di colore del DAC del video vengono copiati sulla tabella il cui indirizzo viene indicato in ES:DX. Il valore di BX (da 00H a 0FFH) indica il primo registro di colore da leggere, e CX contiene il numero di registri coinvolti. La routine del BIOS non effettua alcuna verifica errori; la somma dei valori di BX e CX non deve superare 256 (100H).

La tabella deve contenere tre byte per ogni registro di colore letto. I valori di colore per ogni registro vengono memorizzati sequenzialmente in gruppi di tre byte nella tabella. Il primo byte di ogni gruppo contiene il valore rosso del registro di colore, il secondo il valore verde ed il terzo il valore blu.

AL=18H

Su MCGA e VGA, quando AL=18H, il valore di BL viene copiato nel registro di maschera (3C6H) del DAC del video.

AL=19H

Su MCGA e VGA, quando AL=19H, il valore del registro di maschera (3C6H) del DAC del video viene restituito in BL.

NOTA: Il BIOS dell'adattatore VGA non supporta le sottofunzioni 18H e 19H. Inoltre, la guida tecnica di riferimento dell'interfaccia del BIOS dell'IBM non indica la presenza di queste sottofunzioni, quindi potrebbero non essere supportare nelle future versioni di BIOS.

AL=1AH

Su VGA, quando AL=1AH, i valori correnti del bit 7 del registro di controllo modalità (10H) del controller di attributo ed i bit da 0 a 3 del registro di selezione colore (14H) vengono restituiti rispettivamente in BL e BH. Se il bit 7 del registro di controllo modalità è 1, il valore in BH rappresenta i bit da 0 a 3 del registro di selezione colore. Se il bit 7 del registro di controllo modalità è 0, solo i bit 2 e 3 vengono restituiti come bit 0 e 1 di BH.

AL=1BH

Su MCGA e VGA, quando AL=1BH, viene eseguito il calcolo di scala di grigi di un blocco di registri di colore consecutivi del DAC del video. BX indica il primo registro di colore coinvolto. CX specifica il numero di registri da aggiornare.

L'esempio che segue usa la funzione 10H di INT 10H per aggiornare il valore di colore in un singolo registro di tavolozza:

```
mov         ax,1000h            ; AH := 10H (numero funzione INT 10H)
                                ; AL := 0
mov         bh,6                ; BH := nuovo valore di colore (giallo)
mov         bl,7                ; BL := numero registro di tavolozza
int         10h
```

Per aggiornare il registro di sovrascansione e modificare il colore di cornice visualizzato, richiamate la funzione 10H con AL=1:

```
mov         ax,1001h            ; AH := 10H
                                ; AL := 1
mov         bh,1                ; BH := valore di colore per
                                ;       sovrascansione
int         10h
```

Per caricare tutti i 16 registri di tavolozza ed il registro di sovrascansione da una tabella, richiamate la funzione 10H con AL=2:

```
mov     ax,1002h                ; AH := 10H
                                ; AL := 2
mov     dx,seg PaletteTable
mov     es,dx
mov     dx,offset PaletteTable  ; ES:DX -> tabella di
                                ;  valori di registro di
                                ;  palette
int     10h
```

```
PaletteTable  db      00h,01h,02h,03h,04h,05h,06h,07h
                                                ; registri di tavolozza da 0 a 7
              db      38h,39h,3Ah,3Bh,3Ch,3Dh,3Eh,3Fh
                                      ; registri di tavolozza da 8 a 0FH
              db      00h             ; registro di sovrascansione
```

Per disabilitare l'attributo di lampeggiamento, richiamate la funzione 10H con AL=3 e BL=0:

```
mov           ax,1003h                ; AH := 10H
                                      ; AL := 3
mov           bl,0                    ; BL := 0 (disabilita lampeggiamento)
int           10h
```

Il seguente frammento di programma esegue il calcolo della scala di grigi sui primi 16 registri di colore del DAC del video. I restanti 240 registri restano immutati.

```
mov           ax,101Bh            ; AH := 10H
                                  ; AL := 1BH
mov           bx,0                ; BX := primo registro di colore coinvolto
mov           cx,16               ; CX := numero dei registri di colore
int           10h
```

Funzione 11H: interfaccia generatore caratteri

Registri del chiamante:

AH = 11H

Carica definizioni caratteri alfanumerici.

Tabella definizioni caratteri specificata dall'utente:

AL = 0

BH = punti (byte per definizione carattere)

BL = tabella nella RAM del generatore caratteri

CX = numero di caratteri definiti nella tabella

DX = codice ASCII del primo carattere definito

ES:BP = indirizzo della tabella specificata dall'utente

Definizioni caratteri 8 per 14 del BIOS ROM:

AL = 1

BL = tabella nella RAM del generatore caratteri

Definizioni caratteri 8 per 8 del BIOS ROM:

AL = 2

BL = tabella nella RAM del generatore caratteri

Definizioni caratteri 8 per 16 del BIOS ROM:

AL = 4

BL = tabella nella RAM del generatore caratteri

Seleziona tabelle definizioni caratteri visualizzati.

AL = 3

BL = valore del registro selezione mappa caratteri (EGA,VGA)
= numeri tabella RAM generatore caratteri (MCGA)

Carica definizioni caratteri alfanumerici e programma il controller del CRT.

Tabella di definizioni caratteri specificata dall'utente:

AL = 10H

BH = punti

BL = tabella nella RAM del generatore caratteri

CX = numero di caratteri definiti nella tabella

DX = codice ASCII del primo carattere definito

ES:BP = indirizzo della tabella specificata dall'utente

Definizioni caratteri 8 per 14 del BIOS ROM:

AL = 11H

BL = tabella nella RAM del generatore caratteri

Definizioni caratteri 8 per 8 del BIOS ROM:

AL = 12H

BL = tabella nella RAM del generatore caratteri

Definizioni caratteri 8 per 16 del BIOS ROM:

AL = 14H

BL = tabella nella RAM del generatore caratteri

Carica definizioni caratteri grafici.

Tabella di definizioni caratteri 8x8 specificata dall'utente per vettore di interrupt 1FH:

AL = 20H

ES:BP = indirizzo della tabella di definizioni caratteri specificata dall'utente

Tabella di definizioni caratteri specificata dall'utente:

AL = 21H

BL
= 0 (righe caratteri per schermo specificate in DL)
= 1 14 righe di caratteri per schermo
= 2 25 righe di caratteri per schermo
= 3 43 righe di caratteri per schermo

CX = punti (byte per definizione carattere)

DL = righe caratteri per schermo (quando BL=0)

ES:BP = indirizzo della tabella di definizioni caratteri specificata dall'utente

Definizioni caratteri 8 per 14 del BIOS ROM:

AL = 22H

BL = righe caratteri per schermo (come sopra)

DL = (come sopra)

Definizioni caratteri 8 per 8 del BIOS ROM:

AL = 23H

BL = righe caratteri per schermo (come sopra)

DL = (come sopra)

Definizioni caratteri 8 per 16 del BIOS ROM:

AL = 24H

BL = righe caratteri per schermo (come sopra)

DL = (come sopra)

Rileva informazioni generatore caratteri corrente.

AL = 30H

BH
= 0 Contenuto del vettore di interrupt 1FH
= 1 Contenuto del vettore di interrupt 43H
= 2 Indirizzo della tabella caratteri 8 per 14 ROM
= 3 Indirizzo della tabella caratteri 8 per 8 ROM
= 4 Indirizzo della seconda metà della tabella caratteri 8 per 8 ROM
= 5 Indirizzo della tabella caratteri 9 per 14 ROM alternativa
= 6 Indirizzo della tabella caratteri 8 per 16 ROM
= 7 Indirizzo della tabella caratteri 9 per 16 ROM alternativa

Valori restituiti:

CX = POINTS (altezza della matrice di carattere)

DL = ROWS (righe di caratteri visualizzati -1)

ES:BP = indirizzo della tabella di definizioni caratteri

Aggiornamenti area dati di visualizzazione:

0040:004C	CRT_LEN
0040:0060	CURSOR_MODE
0040:0084	ROWS
0040:0085	POINTS

La funzione 11H di INT 10H comprende una gamma di sottofunzioni che supportano sia i generatori di caratteri alfanumerici sia quelli grafici su EGA, MCGA e VGA. La sottofunzione viene scelta con il valore specificato in AL. Il contenuto degli altri registri varia a seconda della sottofunzione.

AL = 0, 1, 2 o 4

Potete usare le sottofunzioni 0, 1, 2, e 4 per caricare una tabella di definizioni caratteri nella RAM del video in modo che possa essere utilizzata dal generatore di caratteri (il Capitolo 10 descrive in dettaglio questa operazione). Tutte le quattro sottofunzioni sono disponibili su VGA. Su EGA, il BIOS ignora la sottofunzione 4. Il BIOS dell'MCGA non contiene una tabella di definizione caratteri 8 per 14, quindi le chiamate con AL=1 vengono trattate come chiamate con AL=4.

Su MCGA le definizioni caratteri nella RAM del generatore caratteri non vengono visualizzate fino a che non vengono caricate nelle pagine dei tipi di caratteri interne del generatore di caratteri (vedere Capitolo 10). Per realizzare questo tramite il BIOS del video, fate seguire ad ogni chiamata alla funzione 11H eseguita con AL=0, 1, 2 o 4 una chiamata alla funzione 11H con AL=3.

Il CRTC dell'MCGA può visualizzare solo caratteri che presentano un'altezza di 2, 4, 6, 8, 10, 12, 14 o 16 linee. BH deve specificare uno di questi valori. Inoltre, per problemi di compatibilità con il BIOS VGA, la routine del BIOS dell'MCGA estende le definizioni di carattere per i caratteri a 14 linee alle definizioni di caratteri a 16 linee duplicando la quattordicesima linea di ogni definizione carattere.

AL=3

Su EGA e VGA, quando AL=3, la funzione 11H carica il valore indicato in BL nel registro di selezione mappa di carattere del sequencer. Su EGA e MCGA, i bit 0 e 1 di BL indicano quale delle quattro tabelle di 256 caratteri viene utilizzata quando il bit 3 del byte di attributo di un carattere è 0. I bit 2 e 3 di BL indicano quale tabella viene utilizzata quando il bit 3 dell'attributo di un carattere è 1. Su VGA, i bit 0, 1 e 4 specificano una

delle otto tabelle da usarsi quando il bit 3 dell'attributo di un carattere è 0, mentre i bit 2, 3 e 5 specificano la tabella usata quando il bit 3 dell'attributo è 1.

Se entrambi i campi di bit di BL specificano la stessa tabella di definizione caratteri, solo quella tabella viene caricata e visualizzata.

AL=10H, 11H, 12H o 14H

Le sottofunzioni 10H, 11H, 12H e 14H sono analoghe alle sottofunzioni 0, 1, 2 e 4 in quanto caricano una tabella di definizioni caratteri alfanumerici nella RAM del video. La differenza risiede nel fatto che, per queste sottofunzioni su EGA e VGA, il BIOS riprogramma il controller del CRT in modo che possa accettare l'altezza della matrice di carattere. Su MCGA, le chiamate alla funzione 11H con AL=10H, 11H, 12H e 14H sono trattate, rispettivamente, come chiamate alle funzioni 0, 1, 2 e 4.

NOTA: Disabilitate l'emulazione del cursore alfanumerico prima di usare queste sottofunzioni su EGA. La routine di emulazione del cursore del BIOS EGA non produce sempre un cursore alfanumerico soddisfacente (il Capitolo 3 tratta questo argomento in dettaglio).

AL=20H

Se AL=20H, l'indirizzo di ES:BP viene copiato nel vettore di interrupt 1FH a 0000:007C. Questo vettore punta una tabella di definizione caratteri 8 per 8 per i codici ASCII da 80H a FFH. Questa tabella di definizione caratteri viene utilizzata dal BIOS nelle modalità grafiche a 4 colori 320 per 200 e a 2 colori 640 per 200 compatibili CGA.

AL=21H, 22H, 23H o 24H

Le sottofunzioni 21H, 22H, 23H e 24H sono analoghe, rispettivamente, alle sottofunzioni 0, 1, 2 e 4. Il BIOS aggiorna il vettore di interrupt 43H e le variabili POINTS e ROWS dell'area dati di visualizzazione con i valori che descrivono le definizioni di caratteri grafici specificate.

Il BIOS non riprogramma il controller del CRT quando carica le tabelle di definizione caratteri di modalità grafica.

AL=30H

Se AL=30H, la funzione 11H di INT 10H restituisce le informazioni sullo stato corrente del generatore di caratteri. Il valore di POINTS nell'area dati di visualizzazione viene copiato nel registro CX, il valore di ROWS viene ritornato in DL e l'indirizzo di una delle otto tabelle di definizione caratteri viene posto in ES:BP. Il valore di BH indica quale indirizzo di tabella viene restituito.

NOTA: Se richiamate questa sottofunzione su EGA con BH uguale a 6 o 7, o su MCGA con BH uguale a 5 o 7, l'indirizzo ritornato in ES:BP non è definito.

Per selezionare una modalità alfanumerica 80 per 43 su un monitor a 350 linee, richiamate la funzione 11H di INT 10H per caricare il set di caratteri 8 per 8 ROM e riprogrammate il CRTC in modo che visualizzi 43 righe di caratteri (dividendo 350 linee per

8 linee per carattere si ottiene 43 righe di caratteri). Il seguente esempio presume che l'EGA sia già in una modalità alfanumerica 80 per 25 (numero di modalità BIOS 3 o 7).

```
mov                 ax,40h
mov                 es,ax
push                es:[87h                ; mantiene INFO
or                  byte ptr es:[87h],1    ; disabilita emulazione cursore
mov                 ax,1112h          ; AH := 11H (numero funzione INT 10H)
                                      ; AL := 12h (sottofunzione:
                                      ;  carica caratteri alfanumerici 8x8
                                      ;  riprogramma il CRTC)
mov                 bl,0              ; BL := tabella 0 nella RAM del
                                      ;  generatore caratteri
int                 10h
pop                 es:[87h]          ; ripristina INFO
```

Funzione 12H: configurazione sistema di visualizzazione (selezione alternativa)

Registri del chiamante:

AH = 12H

Restituisce informazioni sulla configurazione video:

BL = 10H

Valori restituiti:

BH = modalità video BIOS di default
- 0 Colore
- 1 Monocromatico

BL = quantità di RAM video EGA
- 0 64 KB
- 1 128 KB
- 2 192 KB
- 3 256 KB

CH = bit funzioni

CL = impostazione commutatore di configurazione

Selezione di routine alternativa di stampa schermo:

BL = 20H

Selezione di linee di scansione per le modalità alfanumeriche:

BL = 30H

AL

= 0 200 linee di scansione
= 1 350 linee di scansione
= 2 400 linee di scansione

Valore restituito

AL = 12H

Selezione caricamento tavolozza di default:

BL = 31H

AL

= 0 Abilita caricamento tavolozza di default
= 1 Disabilita caricamento tavolozza di default

Valore restituito:

AL = 12H

Accesso della CPU alla RAM del video:

BL = 32H

AL

= 0 Abilita accesso della CPU alla RAM del video e alle porte di I/O
= 1 Disabilita accesso della CPU alla RAM del video e alle porte di I/O

Valore restituito:

AL = 12H

Calcolo della scala di grigi:

BL = 33H

AL

= 0 Abilita il calcolo della scala di grigi
= 1 Disabilita il calcolo della scala di grigi

Valore restituito:

AL = 12H

Emulazione cursore

BL = 34H

AL

= 0 Abilita emulazione cursore
= 1 Disabilita emulazione cursore

Valore restituito:

5AL = 12H

Commutazione visualizzazione del PS/2:

BL = 35H

AL
- = 0 Adattatore video inizialmente off
- = 1 Video planare inizialmente on
- = 2 Commutatore video attivo off
- = 3 Commutatore video disattivo on

ES:DX = indirizzo dell'area di salvataggio di 128 byte (per AL=0, 2 o 3
Valore restituito:
AL = 12H

Controllo ripristino schermo:

BL = 36H

AL
- = 0 Abilita ripristino
- = 1 Disabilita ripristino

Valore restituito:
AL = 12H

Aggiornamenti area dati di visualizzazione:

(vedere di seguito)

La funzione 12H di INT 10H comprende nove sottofunzioni selezionate tramite il valore contenuto in BL.

BL=10H

Se BL=10H su EGA e VGA, questa routine del BIOS ritorna informazioni sulla configurazione del sistema di visualizzazione. Queste informazioni vengono copiate nell'area dati di visualizzazione da INFO e INFO_3. Queste variabili vengono inizializzate dal BIOS, durante l'avviamento a freddo.

Il valore restituito in BH indica se il sistema di visualizzazione è configurato per una modalità a colori (BH=0) o monocromatica (BH=1). I bit 0 e 1 di BL indicano la quantità di RAM video presente. Dal byte di INFO_3 vengono ricavati i valori ritornati in CH e CL. I bit da 4 a 7 di INFO_3 (input dal connettore funzione EGA) vengono copiati sui bit da 0 a 3 di CH. I bit da 0 a 3 di INFO_3 (impostazioni del commutatore di configurazione) vengono copiati nei bit da 0 a 3 di CL.

BL=20H

Quando BL=20H su MCGA, EGA e VGA il BIOS punta il vettore di interrupt 5 a 0000:0014 su una routine alternativa di stampa schermo contenuta nel BIOS ROM del video. La differenza tra questa routine e la routine del BIOS planare di default è che la versione ROM video usa la variabile ROWS dell'area dati di visualizzazione per determinare il numero di righe di caratteri da stampare. Le versioni di BIOS planare del PC/XT e del PC/AT stampano sempre 25 righe.

BL=30H

Se BL=30H su VGA, la routine del BIOS aggiorna i bit da 0 a 3 del byte INFO_3 (0040:0088) e i bit 7 e 4 del byte di Flags a 0040:0089. La funzione 0 di INT 10H si riferisce a INFO_3 e al byte di Flags per determinare se deve configurare il sistema di visualizzazione per una modalità a 200, a 350 o a 400 linee quando stabilisce una modalità video alfanumerica. Potete così scegliere tra le modalità alfanumeriche a 200, a 350 e a 400 linee eseguendo per prima cosa la funzione 12H di INT 10H con BL=30H e AL=0, 1 o 2, e successivamente richiamando la funzione 0 di INT 10H per impostare la modalità video.

Questa funzione restituisce normalmente il valore 12H in AL. Se la VGA è disabilitata (il bit 3 di INFO è impostato a 1), la funzione restituisce AL=0.

BL=31H

Se BL=31H su MCGA o VGA, la routine del BIOS aggiorna il bit 3 del byte di Flags a 0040:0089 per indicare se devono essere caricati i valori di default della tavolozza del BIOS ROM non appena viene selezionata una nuova modalità video tramite la funzione 0 di INT 10H. Se il valore 0 viene passato ad AL, il bit 3 del byte di Flags viene impostato a 0 per abilitare l'impostazione della tavolozza di default. Se AL=1, il bit 3 viene impostato a 1 per disabilitare l'impostazione della tavolozza di default.

Quando un valore valido viene passato ad AL, la funzione termina con AL=12H.

BL=32H

Quando BL=32H su MCGA o VGA, il valore in AL specifica se l'accesso della CPU al buffer del video e alle porte di I/O è abilitato (AL=0) o disabilitato (AL=1). Sebbene l'interfaccia hardware per il controllo dell'indirizzamento del video si differenzi su MCGA, VGA e adattatore VGA, questa funzione del BIOS è identica in tutti i tre sistemi di visualizzazione (vedere Capitolo 2).

Se un valore valido viene passato in AL, la funzione termina con AL=12H.

NOTA: Sebbene il BIOS del video dell'EGA non supporti questa funzione, potete controllare l'indirizzamento di CPU della RAM del video su EGA aggiornando il bit 1 del registro di output varie (3C2H).

BL=33H

Quando BL=33H su MCGA o VGA, la routine del BIOS aggiorna il bit 1 del byte di Flags a 0040:0089 per indicare se i valori di rosso-verde-blu devono essere ripartiti in va-

lori della scala dei grigi quando le funzioni 0 e 10H di INT 10H aggiornano i registri di colore del DAC del video. Se il valore 0 viene passato in AL, il bit 1 del byte di Flags viene impostato a 1 per abilitare il calcolo della scala dei grigi. Se AL=1, il bit 1 viene impostato a 0 per disabilitare il calcolo della scala dei grigi.

Quando un valore valido viene passato in AL, la funzione termina con AL=12H.

BL=34H

Quando BL=34H su VGA, la routine del BIOS aggiorna il bit 0 di INFO (0040:0087) per indicare se è attiva l'emulazione del cursore del BIOS. Se il valore 0 viene passato in AL, il bit 0 di INFO viene impostato a 0 per abilitare l'emulazione del cursore. Se AL=1, il bit 0 viene impostato a 1 per disabilitare l'emulazione del cursore.

Quando un valore valido viene passato in AL, la funzione termina con AL=12H.

BL=35H

La funzione 1AH di INT 10H con BL=35H fornisce una serie di routine che supportano il passaggio tra due sistemi di visualizzazione PS/2 nello stesso computer. In un computer che contiene due diversi sistemi di visualizzazione compatibili PS/2, le chiamate a questa funzione permettono ad un programma di accedere separatamente al BIOS del video dell'adattatore video e al BIOS del video della piastra madre del PS/2.

Quando accendete un PS/2 che contiene un adattatore video compatibile PS/2, il sistema dell'adattatore è sempre per default il sistema attivo. Per usare il sistema planare (della piastra madre), dovete usare l'interfaccia di commutazione di visualizzazione per disabilitare il sistema dell'adattatore e abilitare il sistema planare.

Potete specificare quattro sottofunzioni correlate alla funzione 12H con BL=35H usando il valore passato nel registro AL. Le quattro sottofunzioni sono progettate per essere chiamate a coppie. Le sottofunzioni 0 e 1 dovrebbero essere chiamate una volta ciascuna per inizializzare l'interfaccia di commutazione di visualizzazione del BIOS e per stabilire una modalità video di default per il sistema di visualizzazione planare. Chiamate successive alle sottofunzioni 2 e 3 vi permetteranno di passare da un sistema di visualizzazione all'altro.

Quando AL=0, il BIOS dell'adattatore inizializza l'interfaccia di commutazione di visualizzazione. Per prima cosa, il BIOS dell'adattatore richiama il BIOS della piastra madre per impostare a 1 il bit 6 del byte di Flags a 0040:0089 per indicare che l'interfaccia è supportata. Successivamente, l'area dati di visualizzazione corrente e i vettori di interrupt video vengono mantenuti nel buffer di 128 byte il cui indirizzo viene passato in ES:DX e i vettori di interrupt video vengono ridiretti al BIOS della piastra madre. Infine, vengono disabilitati il buffer del video dell'adattatore e l'indirizzamento della porta di controllo (vedere funzione 12H di INT 10H, BL=32H).

Quando AL=1, il BIOS della piastra madre stabilisce una modalità alfanumerica 80 per 25 di default su un sistema di visualizzazione planare.

Quando AL=2 e il bit 6 del byte di Flags è 1, il contenuto dell'area dati di visualizzazione e i vettori di interrupt video vengono copiati nel buffer di 128 byte il cui indirizzo

viene passato in ES:DX, ed i vettori di interrupt video vengono ridiretti al BIOS correntemente disattivato. Successivamente il buffer del video e l'indirizzamento della porta di controllo vengono disabilitati per il sistema correntemente attivo. Una chiamata a questa sottofunzione dovrebbe normalmente essere seguita da una chiamata con AL=3.

Quando AL=3 e il bit 6 del byte di Flags è 1, il contenuto dell'area dati di visualizzazione e i vettori di interrupt vengono ripristinati dal buffer il cui indirizzo si trova in ES:DX (questo buffer dovrebbe contenere informazioni precedentemente salvate da una chiamata con AL=0 o AL=2). Successivamente, il buffer del video e l'indirizzamento della porta di controllo vengono abilitati, usando le informazioni video ripristinate.

Quando un valore valido viene passato in AL, e quando sia il BIOS dell'adattatore sia il BIOS planare supportano l'interfaccia di commutazione visualizzazione, ognuna delle quattro sottofunzioni ritorna con AL=12H.

NOTA: Il BIOS del modello 30 del PS/2 (versione 12/12/86 o precedente) e il BIOS del modello 25 del PS/2 (versione 26/6/87) contengono un errore che rende inutilizzabile l'interfaccia di commutazione di visualizzazione. Il problema dovrebbe essere stato risolto nelle successive versioni del BIOS.

BL=36H

Quando BL=36H su VGA, il valore di AL specifica se la routine del BIOS abilita (AL=0) o disabilita (AL=1) il ripristino dello schermo (la disabilitazione temporanea del ripristino dello schermo può accelerare il software che effettua ripetuti accessi alla memoria del video). Il bit 5 del registro di modalità temporizzazione (01H) del sequencer della VGA controlla se il ripristino dello schermo è abilitato o disabilitato. Quando il valore 0 viene passato in AL, il bit 5 viene impostato a 0 per abilitare il ripristino dello schermo; quando AL è 1, il bit 5 viene impostato a 1 per disabilitare il ripristino dello schermo.

La funzione termina sempre con AL=12H.

Per ottenere le informazioni di configurazione EGA, richiamate la funzione 12H di INT 10H con BL=10H:

```
mov                     ah,12h
mov                     bl,10h
int                     10h
```

Per indirizzare la routine alternativa di stampa schermo del BIOS EGA, richiamate la funzione 12H di INT 10H con BL=20H:

```
mov                     ah,12h
mov                     bl,20h
int                     10h
```

Per implementare la commutazione di visualizzazione tra un adattatore VGA e uno MCGA su un modello 30 del PS/2:

```
; salva aree per interfaccia di commutazione visualizzazione
; BIOS del video

VGAsave         db      128 dup(?)          ; salva area per VGA
MCGAsave        db      128 dup(?)          ; salva area per MCGA

; inizializza commutazione visualizzazione
; (esegue questo codice solo una volta)

                mov     ax,1200h            ; AH := 12H
                                            ;  (numero funzione INT 10H)
                                            ; AL := 0
                mov     bl,35h              ; BL := 35H (interfaccia
                                            ;  commutazione visualizzazione)
                mov     dx,seg VGAsave
                mov     es,dx
                mov     dx,offset VGAsave   ; ES:DX -> salva area per info
                                            ;          BIOS VGA
                int     10h
                cmp     al,12h
                jne     Error               ; esce se commutazione
                                            ;  visualizzazione non supportata

                mov     ax,1201h
                mov     bl,35h
                int     10h                 ; disabilita adattatore,
                                            ; abilita video planare

; passa da sistema planare (MCGA) a adattatore (VGA)

                mov     ax,1202h            ; AL := 2 (commutatore video
                                            ;          attivo off)
                mov     bl,35h
                mov     dx,seg VGAsave
                mov     es,dx
                mov     dx,offset VGAsave   ; ES:DX -> salva area per
                                            ;         sistema corrrentemente
                                            ;          attivo
                int     10h
                mov     ax,1203h            ; AL := 3 (commutatore video
                                            ;          disattivato on)
                mov     bl,35h
                mov     dx,offset MCGAsave  ; ES:DX -> salva area per sistema
                                            ;          da rendere attivo
                int     10h

; (per passare da adattatore a planare, scambiare VGAsave e MCGAsave
; nelle chiamate con AL=2 e AL=3)
```

Funzione 13H: visualizzazione stringa di caratteri

Registri del chiamante:

AH = 13H

AL

= 0 BL contiene attributo per la stringa. La posizione del cursore non viene aggiornata.

= 1 BL contiene attributo per la stringa. La posizione del cursore viene aggiornata.

= 2 La stringa contiene byte attributo incorporati. La posizione del cursore non viene aggiornata.

= 3 La stringa contiene byte attributo incorporati. La posizione del cursore viene aggiornata.

BH = pagina video

BL = attributo

CX = lunghezza stringa

DH = riga carattere

DL = colonna carattere

ES:BP = indirizzo di inizio stringa

Valore restituito:

(nessuno)

Aggiornamenti area dati di visualizzazione:

0040:0050 CURSOR_POSN

La funzione 13H di INT 10H scrive una stringa di caratteri nel buffer del video. I caratteri di segnale acustico, di backspace, di avanzamento riga e di ritorno carrello incorporati nella stringa vengono trattati come comandi e non come caratteri visualizzabili. Se la stringa non può essere visualizzata in una riga di caratteri, la funzione 13H fa proseguire la stringa sulla riga successiva. La funzione 13H, inoltre, fa scorrere lo schermo verso l'alto, se necessario.

La stringa viene copiata dall'indirizzo da voi specificato in ES:BP alla locazione del buffer del video indicata dai registri DH e DL (riga e colonna carattere) e dal registro BH (pagina video). Dovete anche specificare il numero di caratteri della stringa nel registro CX.

La funzione 13H comprende quattro sottofunzioni che vengono selezionate a seconda del valore contenuto in AL. Queste quattro sottofunzioni permettono di selezionare il

metodo per la specifica degli attributi di visualizzazione per i caratteri nella stringa e di controllare la posizione finale del cursore dopo che la stringa è stata visualizzata.

Potete specificare l'attributo utilizzato per ogni carattere o in BL (AL=0 o 1) o accoppiando ogni codice di carattere al suo attributo nella stringa stessa (AL=2 o 3). Inoltre, potete indicare se il cursore dovrà restare fermo dopo che la stringa è stata scritta (AL=0 o 2) o se si dovrà spostare sulla posizione di carattere immediatamente successiva alla fine della stringa (AL=1 o 3).

In tutte le modalità grafiche ad eccezione della modalità a 256 colori 320 per 200, l'impostazione a 1 del bit 7 del valore di attributo in BL fa sì che il BIOS effettui l'XOR della stringa nel buffer video.

La pagina video specificata in BH deve essere 0 nella modalità a 4 colori 320 per 200.

NOTA: Su PC/AT, EGA e MCGA, i caratteri di avanzamento riga e ritorno carrello vengono sempre scritti sulla pagina correntemente visualizzata a prescindere dal valore specificato in BH. Se scrivete una stringa contenente uno di questi caratteri di controllo su una pagina video non visualizzata correntemente, la funzione 13H scriverà questi caratteri sulla pagina video sbagliata.

La seguente routine scrive la stringa "Hello, World" nel buffer del video nella pagina video 0 alla riga 12, colonna 34. Per tutti i caratteri della stringa viene usato un valore di attributo 7.

```
                mov     ax,1300h        ; AH := 13H (numero funzione INT 10H)
                                        ; AL := 0 (attributo specificato
                                        ;          in BL, non viene spostato
                                        ;          il cursore)
                mov     bh,0            ; BH := pagina video
                mov     bl,7            ; BL := attributo
                mov     cx,12           ; CX := numero di caratteri da
                                        ;       visualizzare
                mov     dh,12           ; DH := riga 12
                mov     dl,34           ; DL := colonna 34
                mov     bp,seg HelloString
                mov     es,bp
                mov     bp,offset HelloString       ; ES:BP := indirizzo
                                                    ;          stringa
                int     10h


                 .
                 .
                 .
HelloString     db      'Hello, World'
```

Questo esempio visualizza le cifre da 1 a 7 nell'angolo superiore sinistro della pagina video 0. L'attributo usato per ogni cifra corrisponde alla cifra:

```
            mov     ax,1303h    ; AH := 13H (numero funzione INT 10H)
                                ; AL := 3 (la stringa contiene byte di
                                ;  attributo incorporati, sposta il
                                ;  cursore alla fine della stringa)
            mov     bh,0        ; BH := pagina video
            mov     cx,7        ; CX := numero di caratteri da visualizzare
            mov     dx,0        ; DH := riga 0
                                ; DL := colonna 0
            mov     bp,seg StringData
            mov     es,bp
            mov     bp,offset StringData ; ES:BP := indirizzo stringa
            int     10h
             .
             .
             .
StringData  db      '1',1,'2',2,'3',3,'4',4,'5',5,'6',6,'7',7
```

Funzione 14H: (solo per PC convertibile)
Funzione 15H: (solo per PC convertibile)
Funzione 16H: (riservata)
Funzione 17H: (riservata)
Funzione 18H: (riservata)
Funzione 19H: (riservata)
Funzione 1AH: combinazione di visualizzazione

Registri del chiamante:

AH = 1AH

Restituisce combinazione di visualizzazione:

AL = 0

Valori restituiti:

AL = 1AH

BL = visualizzazione abilitata

BH = visualizzazione

Imposta combinazione di visualizzazione:

AL = 1

BL = visualizzazione abilitata

BH = visualizzazione disabilitata

Valore restituito:

AL = 1AH

Aggiornamento area dati di visualizzazione:

0040:008A byte DCC

La funzione 1AH di INT 10H fornisce o aggiorna lo stato di combinazione visualizzazione del BIOS del video. Questo stato viene rappresentato nel byte DCC a 0040:008A nell'area dati di visualizzazione. Questo byte contiene un indice nella tabella di codice di combinazioni di visualizzazione del BIOS ROM, che contiene un elenco di coppie di byte che specificano combinazioni valide di uno o due sistemi di visualizzazione. I sistemi di visualizzazione sono progettati con i seguenti valori.

FFH	Sistema di visualizzazione non riconosciuto
0	Nessun monitor
1	MDA con monitor monocromatico
2	CGA con monitor a colori
3	(riservato)
4	EGA con monitor a colori
5	EGA con monitor monocromatico
6	Controller grafico professionale (P.G.A.)
7	VGA con monitor monocromatico analogico
8	VGA con monitor a colori analogico
9	(riservato)
0AH	MCGA con monitor a colori digitale
0BH	MCGA con monitor monocromatico analogico
0CH	MCGA con monitor a colori analogico

AL=0

Quando AL=0 su MCGA o VGA, la routine del BIOS del video usa il valore del byte DCC come indice nella tabella di codice di combinazioni di visualizzazione e copia l'elemento a due byte della tabella in BH e BL. Se sono presenti due sistemi di visualizzazione, un sistema deve essere monocromatico e l'altro a colori; la routine del BIOS determina quale sistema è attivo esaminando i bit 4 e 5 di EQUIP_FLAG (0040:0010).

AL=1

Quando AL=1 su MCGA o VGA, la routine del BIOS ricerca la combinazione specificata in BH e BL all'interno della tabella di codice di combinazioni di visualizzazione. Se la combinazione specificata viene trovata nella tabella, il byte DCC viene aggiornato con l'indice appropriato nella tabella. Se la combinazione specificata non viene trovata, nel byte DCC viene memorizzato 0FFH.

Quando un valore valido (0 o 1) viene passato in AL, la funzione 1AH di INT 10H ritorna con AL=1AH.

La seguenza riportata di seguito fornisce la combinazione di visualizzazione nei registri BH e BL.

```
mov     ax,1A00h        ; AH := 1AH (numero funzione INT 10H)
                        ; AL := 0
int     10h
cmp     al,1AH
jne     ErrorExit       ; salta se funzione non supportata
                        ; a questo punto BL = visualizzazione
                        ;                     abilitata
                        ; BH := visualizzazione disabilitata
```

Se la sequenza viene eseguita su un modello 30 del PS/2 con un monitor monocromatico analogico collegato all'MCGA ed un monitor monocromatico collegato ad un MDA, i valori ritornati saranno:

AL = 1AH

BL = 0BH (visualizzazione abilitata = MCGA con monitor monocromatico analogico)

BH = 1 (visualizzazione disabilitata = MDA con monitor monocromatico digitale)

Funzione 1BH: funzionalità BIOS del video/informazioni di stato

Registri del chiamante:

AH = 1BH

BX = tipo di implementazione (deve essere 0)

ES:DI = indirizzo del buffer da 64 byte

Valori restituiti:

ES:DI = aggiornamento del buffer con informazioni funzionali e di stato

AL = 1BH

Aggiornamenti area dati di visualizzazione:

(nessuno)

La funzione 1BH di INT 10H fornisce una tabella di informazioni di stato del BIOS del video su MCGA e VGA. La tabella contiene informazioni dinamiche (mostrate nella Figura 146) che vengono determinate quando viene richiamata la funzione 1BH, oltre che informazioni statiche (mostrate nella Figura 147) che descrivono le capacità del BIOS video stesso.

Le informazioni dinamiche vengono copiate nel buffer da 64 byte il cui indirizzo viene passato alla routine del BIOS in ES:DI. L'indirizzo a 32 bit della tabella di informazioni statiche viene fornito come byte da 0 a 3 della tabella di informazioni dinamiche.

Quando viene richiamata con BX=0, la funzione 1BH di INT 10H ritorna sempre con AL=1BH.

Offset	Tipo dati	Descrizione
0	Dword	Indirizzo della tabella di funzionalità statica
4	Byte	Modalità video
5	Word	Numero di colonne di caratteri visualizzati
7	Word	Lunghezza della parte visualizzata del buffer del video in byte
9	Word	Indirizzo iniziale dell'angolo superiore sinistro del buffer del video
0BH	Serie di 16 byte	Tabella locazioni del cursore (colonna, riga) per otto pagine video
1BH	Byte	Linea di cursore fine
1CH	Byte	Linea di cursore inizio
1DH	Byte	Pagina video attiva
1EH	Word	Porta di I/O per registro di indirizzo del CRTC
20H	Byte	CRT_MODE_SET (valore corrente del registro 3x8H)
21H	Byte	CRT_PALETTE (valore corrente del registro 3x9H)
22H	Byte	Numero di righe di caratteri visualizzati
23H	Word	POINTS (altezza della matrice di carattere visualizzato)
25H	Byte	Codice combinazione di visualizzazione abilitata
26H	Byte	Codice combinazione di visualizzazione disabilitata
27H	Word	Numero di colori visualizzati (0 per monocromatico)
29H	Byte	Numero di pagine video supportate
2AH	Byte	Linee scansione del reticolo: 0: 200 linee 1: 350 linee 2: 400 linee 3: 480 linee
2BH	Byte	Tabella caratteri alfanumerici usata quando il bit 3 dell'attributo è 0 (solo VGA)
2CH	Byte	Tabella caratteri alfanumerici usata quando il bit 3 dell'attributo è 1 (solo VGA)
2DH	Byte	Informazioni varie di stato (i bit sono impostati a 1 se lo stato è vero) Bit 0: tutte le modalità attive su tutti i sistemi di visualizzazione (sempre 0 su MCGA) Bit 1: calcolo scala di grigi abilitato Bit 2: monitor monocromatico collegato Bit 3: caricamento tavolozza di default disabilitato

(continua)

Offset	Tipo dati	Descrizione
		Bit 4: emulazione cursore abilitata Bit 5: attributo lampeggiamento abilitato (i bit 6 e 7 sono riservati)
2EH	Byte	(riservato)
2FH	Byte	(riservato)
30H	Byte	(riservato)
31H	Byte	RAM video disponibile 0: 64K 1: 128K 2: 192K 3: 256K
32H	Byte	Stato area salvataggio (i bit sono impostati a 1 se lo stato è vero) Bit 0: due set di caratteri alfanumerici sono attivi (solo VGA) Bit 1: area salvataggio dinamico è attiva Bit 2: aggiramento set di caratteri alfanumerici attivo Bit 3: aggiramento set caratteri grafici attivo Bit 4: aggiramento palette attivo Bit 5: estensione codice combinazione visualizzazione attiva (i bit 6 e 7 sono riservati)
da 33H a 3FH		(riservati)

Figura 146. *Tabella dinamica di stato video fornita dalla funzione 1BH di INT 10H.*

Offset	Tipo dati	Descrizione
0	Byte	Modalità video supportate (bit = 1 se una modalità è supportata) Bit 0: modalità 0 Bit 1: modalità 1 Bit 2: modalità 2 Bit 3: modalità 3 Bit 4: modalità 4 Bit 5: modalità 5 Bit 6: modalità 6 Bit 7: modalità 7
1	Byte	Modalità video supportate (bit = 1 se una modalità è supportata) Bit 0: modalità 8 Bit 1: modalità 9 Bit 2: modalità 0AH

(continua)

Offset	Tipo dati	Descrizione
		Bit 3: modalità 0BH Bit 4: modalità 0CH Bit 5: modalità 0DH Bit 6: modalità 0EH Bit 7: modalità 0FH
2	Byte	Modalità video supportate (bit = 1 se una modalità è supportata) Bit 0: modalità 10H Bit 1: modalità 11H Bit 2: modalità 12H Bit 3: modalità 13H Bit 4: (riservato) Bit 5: (riservato) Bit 6: (riservato) Bit 7: (riservato)
3	Byte	(riservato)
4	Byte	(riservato)
5	Byte	(riservato)
6	Byte	(riservato)
7	Byte	Linee di scansione disponibili nelle modalità alfanumeriche (bit = 1 se supportate) Bit 0: 200 linee Bit 1: 350 linee Bit 2: 400 linee
8	Byte	Numero massimo di set di caratteri alfanumerici visualizzabili
9	Byte	Numero di tabelle di definizione caratteri disponibili nella RAM del generatore di caratteri
0AH	Byte	Funzioni varie del BIOS del video (bit = 1 se disponibili) Bit 0: tutte le modalità su tutti i monitor (funzione 0 di INT 10H) (Nota: questo bit è sempre 0 su MCGA) Bit 1: calcolo della scala di grigi (funzioni 10H e 12H di INT 10H) Bit 2: caricamento del set di caratteri (funzione 11H di INT 10H) Bit 3: caricamento della tavolozza di default (funzione 0 di INT 10H) Bit 4: emulazione cursore (funzione 1 di INT 10H) Bit 5: tavolozza a 64 colori (funzione 10H di INT 10H) Bit 6: caricamento DAC del video (funzione 10H di INT 10H) Bit 7: controllo del DAC del video tramite selezione colore del controller di attributo (funzione 10H di INT 10H)
0BH	Byte	Funzioni varie del BIOS del video (bit = 1 se disponibili) Bit 0: supporto penna ottica (funzione 4 di INT 10H)

(continua)

Offset	Tipo dati	Descrizione
		Bit 1: salva/ripristina stato del video (funzione 1CH di INT 10H)
		Bit 2: lampeggiamento/intensità sfondo (funzione 10H di INT 10H)
		Bit 3: codice combinazioni di visualizzazione (funzione 1AH di INT 10H)
		(i bit da 4 a 7 sono riservati)
0CH	Byte	(riservato)
0DH	Byte	(riservato)
0EH	Byte	Funzioni di salvataggio area
		Bit 0: set di caratteri alfanumerici multipli
		Bit 1: area di salvataggio dinamico
		Bit 2: aggiramento set di caratteri alfanumerici
		Bit 3: aggiramento set di caratteri grafici
		Bit 4: aggiramento tavolozza
		Bit 5: estensione codice combinazioni visualizzazioni (i bit 6 e 7 sono riservati)
0FH	Byte	(riservato)

Figura 147. *Tabella di funzionalità statica. L'indirizzo di questa tabella viene fornito dalla funzione 1BH di INT 10H. La tabella descrive le funzioni del BIOS ROM del sistema di visualizzazione.*

La sequenza riportata qui sotto ritorna le informazioni di stato del BIOS del video nel buffer il cui indirizzo viene passato in ES:DI.

```
            mov     ax,1B00h        ; AH := 1BH (numero funzione INT 10H)
                                    ; AL := 0
            mov     bx,0            ; BX := 0 (tipo di implementazione)
            mov     di,seg StateTable
            mov     es,di
            mov     di,offset StateTable ; ES:DI -> buffer
            int     10h
            cmp     al,1BH
            jne     ErrorExit       ; salta se funzione non supportata
            .
            .                       ; a questo punto StateTable contiene
            .                       ;  la tabella di informazioni
                                    ;  dinamiche
StateTable  db  64  dup(?)
```

Funzione 1CH: salvataggio o ripristino dello stato del video

Registri del chiamante:

AH = 1CH

Ritorna dimensione buffer salvataggio/ripristino:

AL = 0

CX = stati richiesti
Bit 0: stato hardware video
Bit 1: aree dati BIOS del video
Bit 2: stato del DAC del video
Bit da 3 a 0FH: riservati

Valori restituiti:

AL = 1CH

BX = dimensione del buffer in blocchi di 64 byte

Salva stato(i) richiesto(i):

AL = 1

CX = stati richiesti (come sopra)

ES:BX = indirizzo buffer

Ripristino stato(i) richiesto(i):

AL = 2

CX = stati richiesti (come sopra)

ES:BX = indirizzo buffer

Aggiornamenti area dati di visualizzazione:

(vedere sotto)

La funzione 1CH di INT 10H, supportata solo dalla VGA, permette di salvare e ripristinare lo stato dell'hardware del video ed il BIOS ROM del video. La funzione 1CH di INT 10H comprende tre sottofunzioni selezionate dal valore passato in AL. Per ogni sottofunzione, dovete impostare i tre bit meno significativi in CX per indicare la combinazione di stati dei sistemi di visualizzazione che desiderate salvare o ripristinare. Dovete anche passare l'indirizzo di un buffer di salvataggio/ripristino in ES:BX ogni volta che usate la funzione 1CH per salvare o ripristinare lo stato del video.

AL=0

Quando AL=0, la funzione 1CH ritorna la dimensione del buffer richiesto per memorizzare le informazioni di stato per gli stati richiesti in CX. Il valore ritornato in BX è in blocchi di 64 byte.

La funzione 1CH ritorna AL=1CH quando viene richiamata con AL=0 ed almeno uno dei tre bit meno significativi di CX è impostato a 1.

AL=1

Quando AL=1, la funzione 1CH copia le informazioni di stato richieste in CX nel buffer il cui indirizzo è passato in ES:BX.

AL=2

Quando AL=2, la funzione 1CH ripristina lo stato dell'hardware del video, lo stato del BIOS o entrambi utilizzando le informazioni salvate nel buffer il cui indirizzo è passato in ES:BX.

NOTA: La routine del BIOS potrebbe modificare lo stato corrente del video mentre esegue la funzione 1CH. Se pensate di non modificare lo stato del video dopo averlo salvato con la funzione 1CH, ripristinate lo stato del video immediatamente dopo (usando la funzione 1CH con AL=2) per assicurarvi che non venga modificato inavvertitamente.

La sequenza qui riportata opera sotto MS-DOS versione 2.0 o successive. Essa richiama la funzione 48H di INT 21H dell'MS-DOS per allocare la RAM per un buffer di salvataggio/ripristino. Successivamente richiama la funzione 1CH di INT 10H per salvare lo stato corrente del video.

```
mov     ax,1C00h    ; AH := 1CH (numero funzione INT 10H)
                    ; AL := 0
mov     cx,111b     ; CX := 111b (tutti i tre stati del video)
int     10h
cmp     al,1Ch
jne     ErrorExit   ; salta se funzione non supportata
shl     bx,1        ; converte numero di blocchi di 64 byte
shl     bx,1        ;  in numero di blocchi di 16 byte
mov     ah,48h      ; AH := 48H (numero funzione INT 21H MS-DOS)
int     21h         ; AX := segmento di buffer allocato
jc      ErrorExit   ; salta se si verifica errore
mov     es,ax
xor     bx,bx       ; ES:BX -> buffer
mov     cx,111b     ; CX := 111b (tutti i tre stati del video)
mov     ax,1C01h    ; AH := numero funzione INT 10H
                    ; AL := 1
int     10h         ; salva stato del video nel buffer
```

Appendice B
Stampa dello schermo

La maggior parte degli utenti di computer trovano comodo effettuare "un'istantanea" del contenuto corrente dello schermo. Sebbene tutti i membri delle serie di PC e di PS/2 IBM siano dotati di una breve routine di BIOS ROM che invia il contenuto del buffer del video ad una stampante, potreste aver bisogno di scrivere un vostro programma che realizzi "istantanee" dello schermo in aggiunta alla routine ROM. Questa appendice illustra come usare il programma di utilità di stampa schermo del BIOS oltre a spiegare perché e come scrivere una vostra routine a questo scopo.

Modalità alfanumeriche

La routine di stampa schermo alfanumerico della ROM della piastra madre viene richiamata eseguendo l'interrupt software 5 (INT 5) (il gestore della tastiera del BIOS ROM emette questo interrupt quando premete Shift-PrtSc). Questa routine copia il contenuto della pagina video visualizzata correntemente sulla stampante in una modalità alfanumerica 80 per 25 o 40 per 25. La routine stampa solo i codici di carattere ASCII, ignorando i byte di attributo nel buffer del video.

EGA, MCGA, VGA

Il BIOS ROM dell'EGA, dell'MCGA e della VGA contiene una versione più flessibile della routine di stampa schermo di INT 5. Questa versione utilizza il valore ROWS (0040:0084) dell'area dati di visualizzazione per determinare quante righe di caratteri deve stampare (la versione ROM della piastra madre stampa sempre 25 righe). Un PC/XT o un PC/AT IBM utilizza per default la versione della piastra madre. Per rendere accessibile la routine del BIOS ROM EGA o VGA tramite l'interrupt 5, richiamate la funzione 12H di INT 10H con BL=20H. Questo produce il puntamento del vettore di interrupt 5 verso la routine più flessibile.

Caratteri grafici a blocchi

Dal momento che la maggior parte delle stampanti sono progettate per lavorare con molti computer differenti e non solo con i PC IBM, i produttori non sempre progettano le loro stampanti per stampare gli stessi 256 caratteri ASCII che l'hardware del video visualizza nelle modalità alfanumeriche. In particolare, i caratteri usati per la grafica a blocchi non sono sempre disponibili sulle stampanti compatibili per PC. Questi caratteri potrebbero essere stampati in modo diverso da come vengono visualizzati o potrebbero addirittura non essere stampati del tutto.

Modalità grafiche

Il BIOS ROM non supporta le stampe di schermo nelle modalità grafiche, quindi in queste modalità dovete usare un altro programma per stampare il contenuto del buffer del video.

GRAPHICS

GRAPHICS è un programma di stampa schermo in modalità grafica residente in RAM che la Microsoft fornisce come parte dell'MS-DOS sotto il nome di GRAPHICS.COM o GRAPHICS.EXE. Questo programma, quando viene eseguito, stabilisce un programma di stampa schermo residente in memoria per le modalità grafiche CGA (a 4 colori 320 per 200 e a 2 colori 640 per 200). Il programma usa per l'output una stampante a matrice di punti compatibile Epson o IBM.

La parte residente in RAM di GRAPHICS intercetta l'interrupt 5 e verifica la modalità video corrente. Se è attiva una modalità grafica, esegue la stampa di schermo, in caso contrario, la routine di interrupt 5 del BIOS assume il controllo ed esegue la stampa di schermo in modalità alfanumerica. Di conseguenza, una volta eseguito GRAPHICS.COM o GRAPHICS.EXE, potete ottenere una stampa di schermo in modalità grafica premendo Shift-PrtSc, come fareste in una modalità alfanumerica.

Scrittura di una routine di stampa schermo

Se desiderate effettuare istantanee di schermo nelle modalità grafiche EGA, VGA o MCGA originali o su un adattatore Hercules, oppure se GRAPHICS produce un output che non vi soddisfa sulla vostra stampante, potete scrivere una vostra routine di stampa schermo. Il Listato 143 è un esempio di una semplice routine per le modalità grafiche CGA. *ScreenDumpCGA* può essere incorporata in un programma in linguaggio Assembler o in un programma scritto in un linguaggio ad alto livello richiamandola con gli appropriati valori di registri e di modello di memoria (vedere il Capitolo 13 per ulteriori dettagli sull'argomento). Potreste anche costruire *ScreenDumpCGA* all'interno di un programma "concludi ma resta residente" (TSR) che, come GRAPHICS, si collega al vettore di interrupt 5 e viene eseguito ogni volta che viene premuto Shift-PrtSc.

```
                TITLE   'Listato 143'
                NAME    ScreenDumpCGA
                PAGE    55,132

;
; Nome:         ScreenDumpCGA
;
; Funzione:     Stampa di schermo per le modalità a 2 colori 640x200 e a 4
;               colori 320x200 CGA
;
; Chiamante:    (non definito)
;
; Note:         La procedura principale di questo programma, ScreenDumpCGA,
                può essere chiamata da un programma applicativo o come parte
                di un gestore TSR (concludi ma resta residente) per l'interrupt 5.
;

STDPRN          =       4               ; handle standard di stampante MS-DOS


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

;
; PrintLine
;
;       Scrive una linea di caratteri sul dispositivo di stampante
;       standard. Ignora errori.
;

PrintLine       PROC    near            ; Chiamante:  DS:DX - >dati
                                        ;             CX = # di byte
                mov     bx,STDPRN
                mov     ah,40h          ; funzione 40h INT 21h:  scrive
                int     21h
                ret

PrintLine       ENDP

;
; PrinterGraphics
;
;       Attiva la modalità grafica della stampante.  Questa routine deve
;       essere adattata alle diverse stampanti.
;

PrinterGraphics         PROC  9near; Configura stampante Epson MX-80
                                        ; per 480 punti/linea
                mov     dx,offset DGROUP:EpsonGraphics
                mov     cx,3
                call    PrintLine
                ret

PrinterGraphics         ENDP

;
```

```
; PrinterDefault
;
;       Attiva la modalità di default (non grafica) della stampante.
;       Anche in questo caso, questa routine deve essere adattata alle
;       diverse stampanti.
;

PrinterDefaultPROC   near            ; Configura Epson MX-80 per output
                                     ;  alfanumerico di default

             mov     dx,offset DGROUP:EpsonReset
             mov     cx,2
             call    PrintLine
             ret

PrinterDefaultENDP

;
; ChopZeroes
;
;       Elimina zeri di coda dal buffer di output della stampante.
;

ChopZeroes   PROC    near            ; Chiamante: ES:DI -> buffer
                                     ;            CX = lunghezza buffer
                                     ; Ritorna:   CX = lunghezza modificata

             jcxz    L01             ; esce se il buffer è vuoto

             add     di,cx
             dec     di              ; ES:DI -> ultimo byte nel buffer

             xor     al,al           ; AL := 0 (byte da ricercare in scansione)

             std                     ; scansione all'indietro
             repe    scasb
             cld                     ; ripristina flag di direzione
             je      L01             ; salta se il buffer è pieno di zeri

             inc     cx              ; modifica lunghezza oltre ultimo byte non
                                     ;  zero

L01:         ret

ChopZeroes   ENDP

;
; PrintPixels
;
;       Stampa una riga di pixel su un'Epson MX-80.
;

PrintPixels  PROC    near            ; Chiamante:  DI = offset del buffer
                                     ;             CX = lunghezza del buffer

             push    ds
             pop     es              ; ES := DS
```

```
            push    di              ; mantiene offset del buffer
            call    ChopZeroes
            push    cx              ; mantiene lunghezza

            mov     word ptr DataHeader+2,cx   ; memorizza lunghezza del
                                               ;  buffer nell'intesta-
                                               ;  zione dati di output
            mov     dx,offset DGROUP:DataHeader
            mov     cx,4
            call    PrintLine       ; stampa intestazione dati

            pop     cx              ; CX := lunghezza buffer
            pop     dx              ; DX := offset del buffer
            call    PrintLine       ; stampa i pixel

            mov     dx,offset DGROUP:CRLF
            mov     cx,2
            call    PrintLine

            ret

PrintPixels ENDP

;
; TranslatePixels
;
;       Copia una riga stampabile di pixel dal buffer del video al buffer
;       di stampa.Questa routine può essere modificata a piacere per
;       modificare la scalatura o la direzione dell'immagine stampata,
;       per inserire valori di scala di grigi per i pixel a colori, ecc.
;
;       Questa routine formatta il buffer della stampante per l'output su
;       una Epson MX-80.La pagina viene stampata verticalmente, con due
;       pixel stampatio rizzontalmente per ogni pixel verticale nel
;       buffer del video.Dal momento che lo schermo CGA è alto
;       200 pixel, l'output di stampa è largo 400 pixel.

TranslatePixels PROC near           ; Chiamante:  SI = offset del buffer
                                    ;  del video
                                    ;             ES:DI -> buffer di stampa


            push    ds              ; mantiene DS
            mov     ds,VideoBufSeg  ; DS:SI -> buffer del video

            add     di,398          ; ES:DI - 2 byte prima della fine del
                                    ;  buffer

            mov     cx,200          ; CX := # di pixel verticali
            mov     bx,2000h+1      ; BX := primo incremento del buffer
                                    ;  del video
            mov     dx,81-2000h     ; DX := secondo incremento del buffer
                                    ;  del video

            std                     ; riempie il buffer di stampa
                                    ;  all'indietro
L11:        lodsb                   ; AL := 8 pixel dal buffer del video
            mov     ah,al           ; AX := 8 pixel raddoppiati
```

```
                stosw                   ; li scrive sul buffer di stampa

                add     si,bx           ; incremento alla successiva
                xchg    bx,dx           ;  intercalazione del buffer del video

                loop    L11

                cld                     ; azzera flag di direzione
                pop     ds              ; ripristina DS
                ret

TranslatePixels         ENDP

;
; ScreenDumpCGA
;

ScreenDumpCGA PROC      near            ; Chiamante:    DS = DGROUP

                call    PrinterGraphics ; configura la stampante per la grafica

                push    ds
                pop     es              ; DS,ES := DGROUP

                xor     si,si           ; SI := offset dell'inizio del buffer
                                        ;  del video

L21:            push    si
                mov     di,offset DGROUP:PrintBuf
                call    TranslatePixels ; copia una riga stampabile di pixel

                mov     cx,400
                mov     di,offset DGROUP:PrintBuf
                call    PrintPixels     ; li stampa

                pop     si
                inc     si
                cmp     si,80           ; loop di tutte le 80 colonne del
                jb      L21             ;  buffer del video


                call    PrinterDefault  ; riporta la stampante al suo stato
                                        ;  di default
                ret

ScreenDumpCGA ENDP

_TEXT           ENDS


_DATA           SEGMENTword public 'DATA'

PrintBuf        DB      400 dup(?)      ; stampa buffer di output

VideoBufSeg     DW      0B800h

EpsonGraphics DB        1Bh,33h,18h
```

```
EpsonReset      DB      1Bh,40h
DataHeader      DB      1Bh,4Bh,00h,00h
CRLF            DB      0Dh,0Ah

_DATA           ENDS

                END
```

Listato 143. *Una semplice routine di stampa schermo per il CGA.*

ScreenDumpCGA copia i pixel dal buffer del video in un buffer di stampa intermedio. Essa formatta il buffer di stampa in modo che il suo contenuto possa essere inviato direttamente alla stampante (in questo esempio una Epson MX-80). Dal momento che al buffer del video si può accedere casualmente, *ScreenDumpCGA* legge i pixel da esso in un ordine che viene trasmesso alla stampante in modo comodo.

Il fulcro di *ScreenDumpCGA* è rappresentato dalla subroutine *TranslatePixels.* Questa routine mappa i pixel dal buffer del video al buffer della stampante. In questo esempio, la routine è breve e veloce in quanto usa una semplice trasformazione per convertire i pixel di buffer del video in punti della stampante. Dal momento che la Epson MX-80 stampa gruppi di pixel orientati verticalmente (vedere Figura 148), il metodo più semplice per stampare un'immagine dal buffer del video mappato orizzontalmente è quello di ruotarla di 90 gradi.

Per personalizzare *ScreenDumpCGA*, concentratevi su come mappare al meglio i pixel dal buffer del video sulla vostra stampante. Modificate la routine *TranslatePixels* in modo che calcoli in scala o ruoti i pixel in modo diverso, o modificate *ScreenDumpCGA* in modo che cambi l'ordine in cui il contenuto del buffer del video viene copiato sulla stampante.

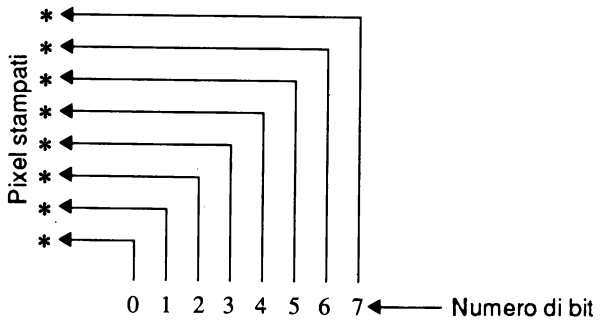


Figura 148. *Mappatura dei pixel per una tipica stampante grafica a matrice di punti. Mentre la testina di stampa si sposta orizzontalmente sulla pagina, stampa otto righe di pixel per volta. Ogni byte di dati trasmesso alla stampante controlla 8 pixel verticali come mostrato nella figura.*

Ad esempio, potreste modificare *ScreenDumpCGA* e *TranslatePixels* per stampare il contenuto del buffer del video dell'EGA o della VGA nella modalità a 16 colori 640 per 350 come nel Listato 144. La routine modificata stampa come punti neri tutti i pixel diversi da zero nel buffer del video. Si noti come la modalità di lettura 1 del controller grafico semplifichi questo compito in *TranslatePixels.*

```
                TITLE   'Listato 144'
                NAME    ScreenDumpEGA
                PAGE    55,132

;
; Nome:         ScreenDumpEGA
;
; Funzione:     Stampa di schermo per la modalità a 16 colori 640x350 EGA
;
; Chiamante:    (non definito)
;
; Note:         La procedura principale di questo programma, ScreenDumpEGA,
;               può essere chiamata  da un programma applicativo o come
;               parte di un gestore TSR (concludi ma resta residente) per
;               l'interrupt 5.
;

STDPRN          =       4               ; handle standard di stampante MS-DOS


DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
                ASSUME cs:_TEXT,ds:DGROUP

;
; PrintLine
;
;       Scrive una linea di caratteri sul dispositivo di stampante
        standard. Ignora errori.
;

PrintLine       PROC    near            ; Chiamante: DS:DX -> dati
                                        ;            CX = # di byte
                mov     bx,STDPRN
                mov     ah,40h          ; funzione 40h INT 21h:  scrive
                int     21h
                ret

PrintLine       ENDP

;
; PrinterGraphics
;
;       Attiva la modalità grafica della stampante. Questa routine deve
        essere adattata alle diverse stampanti.
;
```

```
PrinterGraphics      PROC  near      ; configura stampante Epson MX-80
                                     ;  per 480 punti/linea

             mov     dx,offset DGROUP:EpsonGraphics
             mov     cx,3
             call    PrintLine
             ret

PrinterGraphics  ENDP

;
; PrinterDefault
;
;        Attiva la modalità di default (non grafica) della stampante.
Anche in questo
;        caso questa routine deve essere adattata alle diverse stampanti.

PrinterDefaultPROC   near            ; configura Epson MX-80 per output
                                     ;  alfanumerico di default

             mov     dx,offset DGROUP:EpsonReset
             mov     cx,2
             call    PrintLine
             ret

PrinterDefaultENDP

;
; ChopZeroes
;
;        Elimina zeri di coda dal buffer di output della stampante.
;
ChopZeroes   PROC    near            ; Chiamante: ES:DI -> buffer
                                     ;            CX = lunghezza buffer
                                     ; Ritorna:   CX = lunghezza modificata

             jcxz    L01             ; esce se il buffer è vuoto

             add     di,cx
             dec     di              ; ES:DI -> ultimo byte nel buffer

             xor     al,al           ; AL := 0 (byte da ricercare in
                                     ;          scansione)
             std                     ; scansione all'indietro
             repe    scasb
             cld                     ; ripristina flag di direzione
             je      L01             ; salta se il buffer è pieno di zeri

             inc     cx              ; modifica lunghezza oltre ultimo byte
                                     ;  non zero

L01:         ret

ChopZeroes   ENDP

;
; PrintPixels
;
```

```
;        Stampa una riga di pixel su un'Epson MX-80.
;

PrintPixels   PROC   near           ; Chiamante: DI = offset del buffer
                                    ;            CX = lunghezza del buffer

              push   ds
              pop    es              ; ES := DS

              push   di              ; mantiene offset del buffer
              call   ChopZeroes
              push   cx              ; mantiene lunghezza

              mov    word ptr DataHeader+2,cx   ; memorizza lunghezza
                                                ;  del buffer nell'intesta-
                                                ;  zione dati di output
              mov    dx,offset DGROUP:DataHeader
              mov    cx,4
              call   PrintLine       ; stampa intestazione dati

              pop    cx              ; CX := lunghezza buffer
              pop    dx              ; DX := offset del buffer
              call   PrintLine       ; stampa i pixel

              mov    dx,offset DGROUP:CRLF
              mov    cx,2
              call   PrintLine

              ret

PrintPixels   ENDP



; TranslatePixels
;
;        Copia una riga stampabile di pixel dal buffer del video al
;        buffer di stampa.Questa routine può essere modificata a piacere
;        per modificare la scalatura o la direzione dell'immagine
;        stampata, per inserire valori di scala di grigi per i pixel a
;        colori, ecc.
;
;        Questa routine formatta il buffer della stampante per l'output
;        su una Epson MX-80.
;        La pagina viene stampata verticalmente, quindi l'output di
;        stampa è largo 350 pixel.
;

TranslatePixels PROC near          ; Chiamante: SI = offset del buffer del
                                    ;  video
                                    ;            ES:DI -> buffer di stampa

              push   ds              ; mantiene DS
              mov    ds,VideoBufSeg  ; DS:SI -> buffer del video

              add    di,349          ; ES:DI -> ultimo byte nel buffer di
                                     ;  stampa
```

```
            mov     cx,350          ; CX := # di pixel verticali

; imposta il controller grafico per la modalità di lettura 1

            mov     dx,3CEh         ; porta di I/O del controller grafico
            mov     ax,805h         ; AH := 00001000b (modalità di
                                    ;                  lettura 1)
                                    ; AL := numero registro modalità
            out     dx,ax
            mov     ax,002          ; AH := 0 (valore di confronto colore)
            out     dx,ax           ; AL := numero registro
                                    ;       confronto colore
            mov     ax,0F07h        ; AH := 00001111b (maschera colore
                                    ;  ininfluente)
            out     dx,ax           ; AL := numero registro colore
                                    ;  ininfluente

; riempie il buffer di stampa; tutti i pixel diversi da zero nel buffer
del video vengono stampati

            std                     ; riempie il buffer di stampa
                                    ;  all'indietro

L11:        lodsb                   ; AL := valore di confronto colore a 8
                                    ;  pixel (bit = 0 se pixel<>0)
            not     al              ; AL := 8 pixel stampabili
            stosb                   ; memorizza nel buffer di stampa

add         si,81               ; incrementa alla successiva riga nel
                                    ;  buffer del video
            loop    L11

            cld                     ; azzera flag di direzione

; ripristina stato di default del controller grafico

            mov     ax,5            ; AH := modalità di lettura 0,
                                    ;  modalità di scrittura 0
            out     dx,ax           ; AL := numero registro modalità
            mov     ax,7            ; AH := 0 (maschera di colore
                                    ;          ininfluente)
            out     dx,ax           ; AL := numero registro colore
                                    ;        ininfluente

            pop     ds              ; ripristina DS
            ret

TranslatePixels     ENDP



; ScreenDumpEGA
;

ScreenDumpEGA PROC  near            ; Chiamante:     DS = DGROUP

            call    PrinterGraphics ; configura la stampante per la grafica
```

```
                push    ds
                pop     es              ; DS,ES := DGROUP

                xor     si,si           ; SI := offset dell'inizio del buffer
                                        ;  del video

L21:            push    si
                mov     di,offset DGROUP:PrintBuf
                call    TranslatePixels ; copia una riga stampabile di pixel

                mov     cx,350
                mov     di,offset DGROUP:PrintBuf
                call    PrintPixels     ; li stampa

                pop     si
                inc     si
                cmp     si,80           ; loop su tutte le 80 colonne del
                jb      L21             ;  buffer del video

                call    PrinterDefault  ; riporta la stampante al suo stato
                                        ;  di default
                ret

ScreenDumpEGA   ENDP

_TEXT           ENDS

_DATA           SEGMENTword public 'DATA'

PrintBuf        DB      350 dup(?) ; stampa buffer di output

VideoBufSeg     DW      0A000h

EpsonGraphics   DB      1Bh,33h,18h
EpsonReset      DB      1Bh,40h
DataHeader      DB      1Bh,4Bh,00h,00h
CRLF            DB      0Dh,0Ah

_DATA           ENDS

                END
```

Listato 144. *Una routine di stampa schermo per EGA.*

Definizioni caratteri alfanumerici basati su RAM

Potete anche modificare la routine di stampa schermo in modalità grafica in modo che stampi caratteri basati su RAM usati nelle modalità alfanumeriche su EGA, MCGA, VGA, HGC+ e scheda InColor. La tecnica è quella di utilizzare i codici di carattere memorizzati nella parte visualizzata del buffer del video per indirizzare i pattern di bit nella RAM di definizione caratteri. Il pattern di bit che definisce ogni carattere può quindi essere usato come configurazione di punti per la stampante.

A titolo di esempio, il Listato 145 mostra come procedere per i caratteri definiti nella tabella di definizione caratteri di default nella mappa di memoria 2 su EGA o VGA. La routine stampa ogni colonna di caratteri nel buffer del video riempiendo il buffer (*PrintBuf*) con i pattern di bit che definiscono ogni carattere. L'area di memoria 0 (contenente i codici di carattere) e l'area 2 (contenente le definizioni di carattere) vengono indirizzate separatamente nella subroutine *TranslatePixels* programmando il sequencer e il controller grafico come abbiamo visto nel Capitolo 10.

```
                TITLE   'Listato 145'
                NAME    ScreenDumpAlpha
                PAGE    55,132

;
; Nome:         ScreenDumpAlpha
;
; Funzione:     Stampa di schermo per le modalità alfanumeriche con
;               risoluzione a 350 linee EGA
;
; Chiamante:    (non definito)
;
; Note:         La procedura principale di questo programma, ScreenDumpAlpha,
;               può essere chiamata  da un programma applicativo o come
;               parte di un gestore TSR (concludi ma resta residente)
;               per l'interrupt 5.
;

STDPRN          =       4               ; handle standard di stampante MS-DOS


DGROUP          GROUP   _DATA

_TEXT           SEGMENT  byte public 'CODE'
                ASSUME   cs:_TEXT,ds:DGROUP,es:DGROUP

;
; PrintLine
;
;       Scrive una linea di caratteri sul dispositivo di stampante
;       standard.Ignora errori.
;

PrintLine       PROC    near            ; Chiamante: DS:DX -> dati
                                        ;            CX = # di byte
                mov     bx,STDPRN
                mov     ah,40h          ; funzione 40h INT 21h:  scrive
                int     21h
                ret

PrintLine       ENDP

;
; PrinterGraphics
;
```

```
;       Attiva la modalità grafica della stampante.
;       Questa routine deve essere adattata alle diverse stampanti.
;

PrinterGraphicsn PROC near          ; Configura stampante Epson MX-80
                                    ;  per 480 punti/linea

              mov    dx,offset DGROUP:EpsonGraphics
              mov    cx,3
              call   PrintLine
              ret

PrinterGraphics      ENDP

;
; PrinterDefault
;
;        Attiva la modalità di default (non grafica) della stampante.
;        Anche in questo  caso, questa routine deve essere adattata
;        alle diverse stampanti.
;

PrinterDefaultPROC   near           ; Configura Epson MX-80 per output
                                    ;  alfanumerico di default

              mov    dx,offset DGROUP:EpsonReset
              mov    cx,2
              call   PrintLine
              ret

PrinterDefaultENDP

;
; ChopZeroes
;
;        Elimina zeri di coda dal buffer di output della stampante.
;

ChopZeroes    PROC   near       ; Chiamante: ES:DI -> buffer
                                ;            CX = lunghezza buffer
                                ; Ritorna:   CX = lunghezza modificata

              jcxz   L01        ; esce se il buffer è vuoto

              add    di,cx
              dec    di         ; ES:DI -> ultimo byte nel buffer

              xor    al,al      ; AL := 0 (byte da ricercare in scansione)

              std               ; scansione all'indietro
              repe   scasb
              cld               ; ripristina flag di direzione
              je     L01        ; salta se il buffer è pieno di zeri

              inc    cx         ; modifica lunghezza oltre ultimo byte non
                                ;  zero

L01:          ret
```

```
ChopZeroes    ENDP

;
; PrintPixels
;
;        Stampa una riga di pixel su un'Epson MX-80.
;

PrintPixels   PROC   near           ; Chiamante: DI = offset del buffer
                                    ;            CX = lunghezza del buffer

              push   ds
              pop    es             ; ES := DS

              push   di             ; mantiene offset del buffer
              call   ChopZeroes
              push   cx             ; mantiene lunghezza

              mov    word ptr DataHeader+2,cx   ; memorizza lunghezza del
                                                ;  buffer nell'intestazione
                                                ;  dati di output
              mov    dx,offset DGROUP:DataHeader
              mov    cx,4
              call   PrintLine      ; stampa intestazione dati

              pop    cx             ; CX := lunghezza buffer
              pop    dx             ; DX := offset del buffer
              call   PrintLine      ; stampa i pixel

              mov    dx,offset DGROUP:CRLF
              mov    cx,2
              call   PrintLine

              ret

PrintPixels   ENDP

;
; TranslatePixels
;
;        Copia una riga stampabile di pixel dalla prima tabella di
;        definizioni carattere nella mappa 2 al buffer di stampa.
;
;        Questa routine formatta il buffer della stampante per l'output
;        su una Epson MX-80.La pagina viene stampata verticalmente,
;        quindi l'output di stampa è largo 350 pixel.

TranslatePixels      PROC  near     ; Chiamante: SI = offset del buffer
                                    ;                 video
                                    ;            ES:DI -> buffer di stampa

              push   ds             ; mantiene DS
              mov    ds,VideoBufSeg ; DS:SI -> buffer del video

              add    di,es:PrintBufSize
              dec    di             ; ES:DI -> ultimo byte nel buffer di
                                    ;           stampa
```

```
                mov     dx,3CEh     ; porta di I/O del controller grafico

; riempie il buffer di stampa

                mov     cx,es:Rows  ; CX := numero di righe di caratteri

L11:            push    cx          ; mantiene CX e SI
                push    si

                mov     ax,0004h    ; AH := valore per reg selezione mappa
                                    ;  lettura
                                    ; AL := numero reg selezione mappa lettura
                out     dx,ax       ; seleziona mappa 0 (codici di carattere)

                lodsb               ; AX := successivo codice carattere nel
                                    ;  buffer video
                mov     cl,5
                shl     ax,cl       ; AX := AX * 32
                mov     si,ax       ; SI := offset della definizione carattere
                                    ;  nella mappa 2
                mov     ax,0204h
                out     dx,ax       ; seleziona mappa 2 (configuraz. di bit)

                mov     cx,es:Points
                                    ; CX := dimensione della definiz. di car.
L12:            cld
                lodsb               ; AL := configuraz. a 8 bit dalla tabella
                                    ;  definizione carattere
                                    ; SI := SI + 1
                std
                stosb               ; memorizza configurazione di bit nel
                                    ;  buffer di stampa
                                    ; DI := DI - 1
                loop    L12         ; loop verso il basso sulla definiz. car.

                pop     si          ; ripristina SI e CX
                pop     cx

                add     si,es:Columns
                                    ; DS:SI -> successiva riga di caratteri
                loop    L11         ; loop verso il basso sulle righe di
                                    ;  caratteri
                cld                 ; azzera flag di direzione

                pop     ds          ; ripristina DS
                ret

TranslatePixels         ENDP

;
; ScreenDumpAlpha
;

ScreenDumpAlpha         PROC  near      ;. Chiamante: DS = DGROUP

                call    PrinterGraphics
                                    ; configura la stampante per la grafica
```

```
                call    CGenModeSet; indirizza mappe di memoria EGA in
                                    ;  parallelo: la mappa 0 contiene codici
;                                   ;       di carattere la mappa 2 contiene
;                                   ;       definiz. di carat.

; copia le dimensioni di schermo dall'area dati di visualizzazione

                mov     ax,40h
                mov     es,ax       ; ES -> area dati del BIOS del video

                mov     al,es:[84h]; AX := ROWS
                inc     ax
                mov     Rows,ax
                mov     ax,es:[4Ah]; AX := CRT_COLS
                add     ax,ax       ; * 2 per corretto indirizzam.
                                    ;  del buffer



                mov     Columns,ax
                mov     ax,es:[85h] AX := POINTS
                mov     Points,ax
                mul     Rows        ; AX := ROWS * POINTS
                mov     PrintBufSize,ax

; stampa lo schermo

                push    ds
                pop     es          ; DS,ES := DGROUP
                xor     si,si       ; SI := offset dell'inizio del buffer del
                                        ;  video

L21:            push    si
                mov     di,offset DGROUP:PrintBuf
                call    TranslatePixels; copia una riga stampabile di pixel

                mov     cx,PrintBufSize
                mov     di,offset DGROUP:PrintBuf
                call    PrintPixels      ; li stampa

                pop     si
                add     si,2            ; incrementa a riga di carat. succ.
                cmp     si,Columns      ; loop su tutte le colonne di caratteri
                jb      L21

                call    CGenModeClear   ; ripristina preced. modalità
                                        ;  alfanumerica
                call    PrinterDefault  ; riporta la stampante al suo stato
                                        ;  di default
                ret

ScreenDumpAlpha         ENDP

;
; CGenModeSet (dal Capitolo 10)
;
CGenModeSet     PROC    near
```

```
            push    si          ; mantiene questi registri
            push    cx

            cli                 ; disabilita interrupt
            mov     dx,3C4h     ; indirizzo di porta del sequencer
            mov     si,offset DGROUP:SetSeqParms
            mov     cx,4

L31:        lodsw               ; AH := valore del registro del sequencer
                                ; AL := numero registro
            out     dx,ax       ; programma il registro
            loop    L31
            sti                 ; abilita interrupt

            mov     dl,0CEh     ; DX := 3CEH (indirizzo di porta del
                                ;            controller grafico)


            mov     si,offset DGROUP:SetGCParms
            mov     cx,3

L32:        lodsw               ; programma il controller grafico
            out     dx,ax
            loop    L32

            pop     cx          ; ripristina registri e ritorna
            pop     si
            ret

_CGenModeSet  ENDP

;
; CGenModeClear (dal Capitolo 10)
;

_CGenModeClearPROC  near

            push    si          ; mantiene questi registri
            push    cx

            cli                 ; disabilita interrupt
            mov     dx,3C4h     ; indirizzo di porta del sequencer
            mov     si,offset DGROUP:ClearSeqParms
            mov     cx,4



L41:        lodsw               ; AH := valore del registro del sequencer
                                ; AL := numero registro
            out     dx,ax       ; programma il registro
            loop    L41
            sti                 ; abilita interrupt

            mov     dl,0CEh     ; DX := 3CEH (indirizzo di porta del
                                ;            controller grafico)
            mov     si,offset DGROUP:ClearGCParms
```

```
            mov     cx,3

L42:        lodsw               ; programma controller grafico
            out     dx,ax
            loop    L42

            mov     ah,0Fh      ; AH := numero funzione INT 10H
            int     10h         ; attiva modalità video

            cmp     al,7
            jne     L43         ; salta se non è modalità monocromatica

            mov     ax,0806h    ; programma controller grafico
            out     dx,ax       ;  per iniziare mappa a B000:0000

L43:        pop     cx          ; ripristina registri e ritorna
            pop     si
            ret

_CGenModeClearENDP

_TEXT       ENDS


_DATA       SEGMENT   word public 'DATA'

PrintBuf    DB      400 dup(?) ; stampa buffer di output

VideoBufSeg DW      0A000h

EpsonGraphics DB    1Bh,33h,18h
EpsonReset  DB      1Bh,40h
DataHeader  DB      1Bh,4Bh,00h,00h
CRLF        DB      0Dh,0Ah

Columns     DW      ?           ; numero colonne
                                ;  di caratteri visualizzati
Rows        DW      ?           ; numero righe
                                ;  di caratteri visualizzati
Points      DW      ?           ; dimensione verticale
                                ;  matrice di carat.
PrintBufSize DW     ?           ; righe * punti

SetSeqParms DW      0100h       ; parametri per CGenModeSet
            DW      0402h
            DW      0704h
            DW      0300h

SetGCParms  DW      0204h
            DW      0005h
            DW      0006h

ClearSeqParms DW    0100h       ; parametri per CGenModeClear
            DW      0302h
            DW      0304h
            DW      0300h
```

```
ClearGCParms  DW      0004h
              DW      1005h
              DW      0E06h

_DATA         ENDS

              END
```

Listato 145. *Uso di tabelle di definizione caratteri basati su RAM per stampare il set di caratteri.*

Appendice C
Identificazione dei sistemi di visualizzazione

Sono due i motivi per cui i programmi devono determinare la configurazione dello hardware di visualizzazione sulla quale operano. Una ragione è per mantenere la trasportabilità. Un programma che riconosce il sistema di visualizzazione del computer sul quale opera può adattarsi a specifiche configurazioni hardware. Immaginate, ad esempio, un programma che visualizzi sia testo sia immagini grafiche. Questo programma potrebbe visualizzare il testo e la grafica su uno stesso schermo di un computer che è dotato di un solo sistema di visualizzazione, ma potrebbe anche sfruttare una configurazione a doppio sistema di visualizzazione introducendo il testo in uno schermo e la grafica in un altro.

Un altro motivo per cui è importante far sì che l'ambiente hardware venga esaminato dal programma è per permettere allo stesso programma di utilizzare le routine di output su video più veloci possibili. Ad esempio, se il vostro programma opera in una modalità alfanumerica su CGA, potreste dover eliminare l'effetto neve effettuando la sincronizzazione con i segnali di temporizzazione del controller del CRT. Questo compito di servizio può però essere evitato se il programma sta operando su un altro sistema di visualizzazione. Se il vostro programma "sa" che non sta operando su CGA, può utilizzare routine di output su video più veloci che non eseguono il lavoro di servizio per evitare l'effetto neve.

CGA e cloni

Purtroppo, per gli adattatori grafici a colori (CGA) e per i cloni non esiste alcun metodo affidabile per determinare se l'hardware gestisce i conflitti che si verificano negli accessi alla memoria del buffer del video senza interferenze con il video (vedere Capitolo 3). Se il vostro programma deve operare su CGA, potreste dover chiedere all'utente di configurare le vostre routine di output alfanumerico verificando se producono o meno l'effetto neve.

Potete anche rilevare se il programma sta operando su un clone CGA che non presenta il problema dell'effetto neve nelle modalità alfanumeriche. Se sapete che il vostro programma opererà anche su un clone CGA come quello incorporato in un COMPAQ o in un AT&T 6300, potete ricercare nel BIOS ROM una stringa che indichi il nome del computer, ad esempio, "COMPAQ". Potreste anche esaminare il byte ID del BIOS ROM a F000:FFFE per determinare se il vostro programma sta operando su un membro della famiglia di PC IBM che non presenta il problema dell'effetto neve (come ad esempio il PCjr).

Altri adattatori video

Sebbene la determinazione del fatto che un particolare CGA o clone presenti il problema dell'effetto neve nelle modalità alfanumeriche possa essere un compito arduo, la

distinzione tra i vari comuni adattatori video IBM è invece relativamente facile. Alcune delle tecniche descritte in questa appendice si basano su peculiarità del firmware o dell'hardware dei diversi adattatori, ma tutte sono basate sulle raccomandazioni IBM ed Hercules.

PS/2

Sui PS/2, la funzione 1AH di INT 10H permette di determinare quale sistema di visualizzazione è presente ed attivo nel computer (vedere Appendice A). Naturalmente, il BIOS del video del PS/2 non riconosce gli adattatori video non IBM. Ad esempio, se utilizzate un adattatore Hercules in un modello 30 del PS/2, una chiamata alla funzione 1AH di INT 10H ritorna solo l'informazione che un adattatore compatibile MDA è presente nel sistema. L'identificazione dell'adattatore a questo punto dipende solo da voi.

VideoID, la routine del Listato 146, rileva la presenza di uno o due adattatori. Se sono presenti due adattatori, *VideoID* indica quale adattatore è attivo (cioè quale adattatore il BIOS sta correntemente utilizzando per l'output). Le tecniche usate per identificare ogni adattatore sono descritte nel listato.

```
                TITLE   'Listato 146'
                NAME    VideoID
                PAGE    55,132

;
; Nome:         VideoID
;
; Funzione:     Rileva la presenza dei vari sistemi di visualizzazione
;               e monitor collegati.
;
; Chiamante:    Microsoft C:
;
;                       azzera VideoID(VIDstruct);
;
;                       struct
;                       {
;                         char VideoSubsystem;
;                         char Display;
;                       }
;                                 *VIDstruct[2];
;
;               valori ID sistema:
;                                  0  = (nessuno)
;                                  1  = MDA
;                                  2  = CGA
;                                  3  = EGA
;                                  4  = MCGA
;                                  5  = VGA
```

```
;                                   80h = HGC
;                                   81h = HGC+
;                                   82h = scheda InColor Hercules
;
;               tipi di monitor:    0  = (nessuno)
;                                   1  = monocromatico compatibile MDA
;                                   2  = a colori compatibile CGA
;                                   3  = a colori compatibile EGA
;                                   4  = monocromatico compatibile PS/2
;                                   5  = a colori compatibile PS/2
;
;
;           I valori forniti da VIDstruct[0].VideoSubsystem e
;           VIDstruct[0].Display indicano il sistema attivo correntemente.
;

ARGpVID         EQU     word ptr [bp+4]  ; indirizzamento cornice di stack


VIDstruct       STRUC                    ; corrisponde a struttura dati in C

Video0Type      DB      ?                ; primo tipo di sistema
Display0Type    DB      ?                ; monitor collegato al primo sistema

Video1Type      DB      ?                ; secondo tipo di sistema
Display1Type    DB      ?                ; monitor collegato al secondo sistema

VIDstruct       ENDS


Device0         EQU     word ptr Video0Type[di]
Device1         EQU     word ptr Video1Type[di]


MDA             EQU     1                ; tipi di sistemi
CGA             EQU     2
EGA             EQU     3
MCGA            EQU     4
VGA             EQU     5
HGC             EQU     80h
HGCPlus         EQU     81h
InColor         EQU     82h

MDADisplay      EQU     1                ; tipi di monitor
CGADisplay      EQU     2
EGAColor Display        EQU         3
PS2Mono Display         EQU         4
PS2Color Display        EQU         5

TRUE            EQU     1
FALSE           EQU     0



DGROUP          GROUP   _DATA

_TEXT           SEGMENTbyte public 'CODE'
```

```
                ASSUME cs:_TEXT,ds:DGROUP

                PUBLIC _VideoID
_VideoID        PROC   near

                push   bp              ; mantiene registri del chiamante
                mov    bp,sp
                push   si
                push   di

; inizializza la struttura dati che conterrà i risultati

                mov    di,ARGpVID      ; DS:DI -> inizio della struttura dati

                mov    Device0,0       ; azzera queste variabili
                mov    Device1,0

; ricerca i vari sistemi usando le subroutine i cui indirizzi sono
;  tabulati in TestSequence; ogni subroutine imposta flag in TestSequence
;  per indicare se subroutine successive devono essere richiamate

                mov    byte ptr CGAflag,TRUE
                mov    byte ptr EGAflag,TRUE
                mov    byte ptr Monoflag,TRUE

                mov    cx,NumberOfTests
                mov    si,offset DGROUP:TestSequence

L01:            lodsb                  ; AL := flag
                test   al,al
                lodsw                  ; AX := indirizzo di subroutine
                jz     L02             ; salta subroutine se flag è falsa

                push   si
                push   cx
                call   ax              ; richiama subroutine per rilevare
                                       ;  sistema
                pop    cx
                pop    si

L02:            loop   L01

; determina quale sistema è attivo

                call   FindActive

                pop    di              ; ripristina registri del chiamante e
                                       ;  ritorna
                pop    si
                mov    sp,bp
                pop    bp
                ret
```

```
_VideoID        ENDP


;
; FindPS2
;
;         Questa subroutine usa la funzione 1Ah di INT 10H
;         per determinare il codice di combinazione di
;         visualizzazione (DCC) del BIOS video per ogni sistema
;         di visualizzazione presente.
;

FindPS2         PROC    near

                mov     ax,1A00h
                int     10h             ; richiama il BIOS del video per info

                cmp     al,1Ah
                jne     L13             ; esce se funzione non supportata
                                        ;  (cioè né MCGA né VGA sono nel sistema)

; converte DCC del BIOS in sottosistemi e monitor specifici

                mov     cx,bx
                xor     bh,bh           ; BX := DCC per sistema attivo

                or      ch,ch
                jz      L11             ; salta se solo un sistema è presente

                mov     bl,ch           ; BX := DCC inattivo
                add     bx,bx
                mov     ax,[bx+offset DGROUP:DCCtable]

                mov     Device1,ax

                mov     bl,cl
                xor     bh,bh           ; BX := DCC attivo

L11:            add     bx,bx
                mov     ax,[bx+offset DGROUP:DCCtable]

                mov     Device0,ax

; azzera flag per sistemi che sono stati esclusi

                mov     byte ptr CGAflag,FALSE
                mov     byte ptr EGAflag,FALSE
                mov     byte ptr Monoflag,FALSE

                lea     bx,Video0Type[di]     ; se il BIOS ha riportato un MDA ...
                cmp     byte ptr [bx],MDA
                je      L12

                lea     bx,Video1Type[di]
                cmp     byte ptr [bx],MDA

                jne     L13
```

```
L12:            mov     word ptr [bx],0 ; ... Hercules non può essere escluso
                mov     byte ptr Monoflag,TRUE

L13:            ret

FindPS2         ENDP

;
; FindEGA
;
; Ricerca un EGA.  Questo viene realizzato effettuando una chiamata ad
; una funzione del BIOS dell'EGA che non esiste nel BIOS
; di default (MDA, CGA).

FindEGA         PROC    near            ; Chiamante: AH = flag
                                        ; Ritorna:   AH = flag
                                        ;        Video0Type e
                                        ;        Display0Type aggiornati
                mov     bl,10h          ; BL := 10h (ritorna info su EGA)
                mov     ah,12h          ; AH := numero funzione INT 10H
                int     10h             ; richiama BIOS EGA per info
                                        ; se BIOS EGA è presente,
                                        ;  BL <>10H
                                        ;  CL = impostazione commutatore
                cmp     bl,10h
                je      L22             ; salta se BIOS EGA non è presente

                mov     al,cl
                shr     al,1            ; AL := commutatori/2
                mov     bx,offset DGROUP:EGADisplays
                xlat                    ; determina tipo monitor da commutatori
                mov     ah,al           ; AH := tipo monitor
                mov     al,EGA          ; AL := tipo sistema
                call    FoundDevice

                cmp     ah,MDADisplay
                je      L21             ; salta se EGA ha un monitor
                                        ;  monocromatico

                mov     CGAflag,FALSE   ; non è CGA se EGA ha monitor a colori
                jmp     short L22

L21:            mov     Monoflag,FALSE  ; EGA ha un monitor
                                        ;  monocromatico, quindi MDA
                                        ;  ed Hercules sono esclusi
L22:            ret

FindEGA         ENDP


;
; FindCGA
;
;       Questo viene realizzato cercando il CRTC 6845 del CGA alla porta
;       di I/O 3D4H.
;
FindCGA         PROC    near            ; Ritorna: VIDstruct aggiornato
```

```
                mov     dx,3D4h         ; DX := porta indirizzo CRTC
                call    Find6845
                jc      L31             ; salta se non presente

                mov     al,CGA
                mov     ah,CGADisplay
                call    FoundDevice

L31:            ret

FindCGA         ENDP


;
; FindMono
;
;       Questo viene realizzato cercando il CRTC 6845 dell'MDA
;       alla porta di I/O 3B4H.
;       Se un 6845 viene trovato,
;       la subroutine distingue tra un adattatore MDA
;       ed un adattatore Hercules verificando il bit 7
;       del byte di stato del CRT.
;       Questo bit cambia su adattatori Hercules ma non cambia su MDA.

;       I vari adattatori Hercules vengono identificati
;       dai bit da 4 a 6 del valore di stato del CRT:
;
;               000b = HGC
;               001b = HGC+
;               101b = scheda InColor
;

FindMono        PROC    near            ; Ritorna:  VIDstruct aggiornato

                mov     dx,3B4h         ; DX := porta indirizzo CRTC
                call    Find6845
                jc      L44             ; salta se non presente

                mov     dl,0BAh         ; DX := 3BAh (porta di stato)
                in      al,dx
                and     al,80h
                mov     ah,al           ; AH := bit 7 (sincronizzazione
                                          verticale su HGC)

                mov     cx,8000h        ; ripete questo 32768 volte
L41:            in      al,dx
                and     al,80h          ; isola bit 7
                cmp     ah,al
                loope   L41             ; attende modifica bit 7


                jne     L42             ; se bit 7 modificato, è un Hercules

                mov     al,MDA          ; se bit 7 non cambia, è un MDA
                mov     ah,MDADisplay
                call    FoundDevice
                jmp     short L44
```

```
L42:            in      al,dx
                mov     dl,al           ; DL := valore da porta di stato

                mov     ah,MDADisplay   ; presume sia un monitor monocromatico

                mov     al,HGC          ; ricerca un HGC
                and     dl,01110000b    ; maschera bit da 4 a 6
                jz      L43

                mov     al,HGCPlus      ; ricerca un HGC+
                cmp     dl,00010000b
                je      L43             ; salta se è un HGC+

                mov     al,InColor      ; è una scheda InColor
                mov     ah,EGAColorDisplay

L43:            call    FoundDevice

L44:            ret

FindMono        ENDP


;
; Find6845
;
;       Questa routine rileva la presenza del CRTC su un MDA,
;       CGA o HGC. La tecnica è quella di scrivere e di leggere
;       il registro 0Fh del chip (cursore basso).
;       Se lo stesso valore viene letto come scritto,
;       si assume che il chip sia presente all'indirizzo
;       di porta specificato.
;

Find6845        PROC    near            ; Chiamante:DX = indirizzo di porta
                                        ; Ritorna:  cf impostato se non presente
                mov     al,0Fh
                out     dx,al           ; seleziona reg 0Fh
                                        ;  (cursore basso) 6845
                inc     dx
                in      al,dx           ; AL := valore corrente
                                        ;  di cursore basso
                mov     ah,al           ; mantiene in AH
                mov     al,66h          ; AL := valore arbitrario
                out     dx,al           ; tenta di scrivere su 6845

                mov     cx,100h

L51:            loop    L51                     ; attende che 6845 risponda

                in      al,dx
                xchg    ah,al                   ; AH := valore ritornato
                                                ; AL := valore originale
                out     dx,al                   ; ripristina valore originale

                cmp     ah,66h                  ; verifica che 6845
                                                ;  abbia risposto
                je      L52                     ; salta in caso affermativo
```

```
                                        ;  (cf viene azzerato)

            stc                         ; imposta flag riporto se
                                        ;  6845 non è presente

L52:        ret

Find6845    ENDP


;
; FindActive
;
;       Questa subroutine memorizza il dispositivo correntemente
;       attivo come Device0.La modalità video corrente determina
;       quale sistema è attivo.
;

FindActive  PROC    near

            cmp     word ptr Device1,0
            je      L63                         ; esce se è un solo sistema

            cmp     Video0Type[di],4            ; esce se MCGA
                                               ;   o VGA presente
            jge     L63                         ; (funzione 1AH di INT
                                                ;  10H
            cmp     Video1Type[di],4            ;  ha già svolto il compito)
            jge     L63

            mov     ah,0Fh
            int     10h                         ; AL := modalità video
                                                ;  BIOS corrente

            and     al,7
            cmp     al,7                        ; salta se monocromatico
            je      L61                         ;  (modalità 7 o 0Fh)

            cmp     Display0Type[di],MDADisplay
            jne     L63                        ; esce se Display0 è a colori
            jmp     short L62

L61:        cmp     Display0Type[di],MDADisplay
            je      L63                         ; esce se Display0 è
                                                ;  monocromatico


L62:        mov     ax,Device0                  ; rende correntemente
                                                ;  attivo Device0
            xchg    ax,Device1
            mov     Device0,ax



L63:        ret

FindActive  ENDP
```

```
;
; FoundDevice
;
;        Questa routine aggiorna l'elenco dei sottosistemi.
;

FoundDevice   PROC   near                 ; Chiamante:AH = monitor #
                                          ;           AL = sottosistema #
                                          ; Distrugge:BX
              lea    bx,Video0Type[di]
              cmp    byte ptr [bx],0
              je     L71                  ; salta se primo sistema

              lea    bx,Video1Type[di]    ; deve essere secondo sistema

L71:          mov    [bx],ax              ; aggiorna elemento di elenco
              ret

FoundDevice   ENDP

_TEXT         ENDS



_DATA         SEGMENT   word public 'DATA'

EGADisplays   DB     CGADisplay           ; 0000b, 0001b(valori commutatore
                                          ;               EGA)
              DB     EGAColorDisplay      ; 0010b, 0011b
              DB     MDADisplay           ; 0100b, 0101b
              DB     CGADisplay           ; 0110b, 0111b
              DB     EGAColorDisplay      ; 1000b, 1001b
              DB     MDADisplay           ; 1010b, 1011b

DCCtable      DB     0,0                  ; traduce tabella per funzione
                                          ;  1Ah di INT 10h
              DB     MDA,MDADisplay
              DB     CGA,CGADisplay
              DB     0,0
              DB     EGA,EGAColorDisplay
              DB     EGA,MDADisplay
              DB     0,0
              DB     VGA,PS2MonoDisplay
              DB     VGA,PS2ColorDisplay
              DB     0,0
              DB     MCGA,EGAColorDisplay
              DB     MCGA,PS2MonoDisplay
              DB     MCGA,PS2ColorDisplay

TestSequence  DB     TRUE                 ; questo elenco di flag
                                          ;  e di indirizzi
              DW     FindPS2              ;  determina l'ordine nel quale
                                          ;   questo programma cerca i vari
EGAflag       DB     ?                    ;  sistemi
              DW     FindEGA
```

```
CGAflag        DB      ?
               DW      FindCGA

Monoflag       DB      ?
               DW      FindMono

NumberOfTests  EQU     ($-TestSequence)/3

_DATA          ENDS

               END
```

Listato 146. *Una routine per identificare i sistemi di visualizzazione dei PC e dei PS/2.*

La routine *VideoID* controlla gli adattatori procedendo per eliminazione. Ad esempio, se la routine viene eseguita su un PS/2, la chiamata a INT 10H fornisce le informazioni desiderate. Su PC/XT e PC/AT, se viene rilevato un EGA con un monitor monocromatico collegato, non c'è motivo di ricercare un MDA o una scheda Hercules nello stesso sistema. Se è presente un adattatore monocromatico, la routine differenzia tra MDA ed i vari adattatori Hercules.

CONSIGLIO

La funzione 1AH di INT 10H sull'adattatore VGA non riporta la presenza dell'MCGA quando l'adattatore è installato in un modello 30 del PS/2. Inoltre, la funzione 1AH nell'MCGA ignora la presenza di un EGA se questo è installato in un modello 30. Se queste combinazioni vi interessano, dovrete verificarle esplicitamente dopo aver richiamato la funzione 1AH di INT 10H (nella prima situazione, esaminate il byte di indentificazione del BIOS della piastra madre a F000:FFFE per rilevare la presenza di un modello 30. Nel secondo caso, eseguite la funzione 12H di INT 10H con BL=10H per rilevare la presenza di un EGA).

Il programma in C del Listato 147 mostra come utilizzare *VideoID*.

Glossario

Questo glossario comprende alcuni acronimi, abbreviazioni, termini complicati, termini tecnici e di gergo di programmazione che appaiono frequentemente nel corso del libro.

80x86: Si riferisce a tutti i processori della famiglia Intel 8086. I PC ed i PS/2 IBM utilizzano tutti uno di questi processori: 8086, 8088, 80286 o 80386.

adattatore: Un circuito modulare a connettore che svolge il compito specializzato di generare output su video. I famosi adattatori video dei PC IBM comprendono MDA, CGA, HGC, EGA e adattatore VGA.

adattatore VGA: L'adattatore di visualizzazione dei PS/2 IBM.

Adattatore Video PS/2: Un adattatore video IBM compatibile VGA che può essere utilizzato su PC/XT, PC/AT o PS/2 modello 30; comunemente chiamato "adattatore VGA".

ANSI: American National Standards Institute (istituto nazionale americano per gli standard). Una delle molte attività dell'ANSI è quella di garantire la standardizzazione degli strumenti di programmazione, tra cui i linguaggi (come ad esempio C e FORTRAN) e le interfacce software (come GKS).

APA: All Points Addressable (tutti i punti indirizzabili); descrive le modalità grafiche su CGA, EGA e schede grafiche Hercules.

API: Application Program Interface (interfaccia per programma applicativo); una serie di routine a livello sistema che può essere utilizzata in un programma applicativo per l'input e l'output di base, la gestione dei file e così via. In un ambiente operativo orientato alla grafica come Windows della Microsoft, il supporto ad alto livello per l'output grafico su video fa parte dell'API.

area dati di controllo video: Parte dell'area dati di visualizzazione. Il blocco di RAM da 0040:0049 a 0040:0066 è l'area dati di controllo video 1; il blocco tra 0040:0084 e 0040:008A è l'area dati di controllo video 2.

area dati di visualizzazione: Un'area globale di dati mantenuta dal BIOS ROM per la memorizzazione di parametri correlati alle sue routine di I/O video INT 10H.

ASCII: American Standard Code for Information Interchange (codice standard americano per lo scambio delle informazioni). Lo standard ASCII specifica il set di caratteri di base usato nei PC e nei PS/2 IBM.

attributi: Colore, intensità, lampeggiamento ed altre caratteristiche visualizzate dei caratteri o dei pixel.

BIOS: Basic Input/Output System (sistema di input/output di base); un'interfaccia di programmazione a basso livello con i principali dispositivi di I/O del sistema.

BIOS planare: Routine del BIOS che si trovano in ROM sulla piastra madre del PC e dei PS/2.

buffer del video: Un buffer che contiene i dati che appaiono sullo schermo; conosciuto sotto i nomi di "buffer di visualizzazione", "buffer di quadro", "buffer di ripristino" o "buffer rigenerativo".

CGA: Color Graphics Adapter (adattatore grafico a colori) dell'IBM.

codice di carattere: Un codice numerico associato ad un carattere. Il set di caratteri ASCII di default utilizzato in tutti i PC e i PS/2 comprende 256 codici di carattere a 8 bit.

CPU: Central Processing Unit (unità centrale di elaborazione), o processore principale all'interno di un computer. Ad esempio, la CPU in un PC è un Intel 8088 e in un PC/AT è un 80286.

CRT: Cathode Ray Tube (tubo a raggio catodico), o lo schermo che vedete quando guardate un monitor di un computer. Alcune persone si riferiscono all'intero monitor (lo schermo e la circuiteria associata) come ad un CRT.

CRTC: Controller del CRT; un chip che controlla i segnali di temporizzazione di un monitor.

definizione: Il processo che determina quale area di un'immagine grafica si trova entro confini specificati.

DGIS: Direct Graphics Interface Specification (specifiche di interfaccia grafica diretta); un'interfaccia grafica firmware progettata per i sistemi di visualizzazione basati su coprocessori grafici hardware.

driver: Software o firmware che programma direttamente un dispositivo hardware specifico come ad esempio un adattatore video o una stampante.

EBCDIC: Extended Binary Coded Decimal Interchange Code (codice di interscambio decimale codificato binario esteso); l'implementazione di set di caratteri usata sui computer mainframe dell'IBM.

EGA: Enhanced Graphics Adapter (adattatore grafico evoluto).

gate array: Un circuito integrato che viene parzialmente preassemblato in fabbrica. Un circuito integrato specifico per un'applicazione basato sulla tecnologia gate array può essere meno costoso e può essere fabbricato più rapidamente di un circuito integrato custom.

GKS: Graphical Kernel System (sistema di base grafico); un'interfaccia grafica standard ad alto livello.

HGC: Scheda grafica monocromatica Hercules.

HGC+ (HGC Plus): Scheda grafica Plus Hercules; un adattatore video monocromatico come l'HGC, ma con un generatore di caratteri hardware che può usare set di caratteri basati su RAM.

InColor: Scheda InColor Hercules; una versione a 16 colori della HGC+.

latch: Un registro hardware esterno alla CPU che viene utilizzato per memorizzare temporaneamente dei dati. Ad esempio, il controller grafico dell'EGA usa quattro latch interni ad 8 bit per mediare i trasferimenti di dati tra i piani di bit e la CPU.

linea di scansione: Una linea orizzontale tracciata sullo schermo dal raggio elettronico del CRT.

LSI: Large Scale Integration (integrazione a larga scala).

matrice di carattere: La matrice rettangolare di pixel all'interno della quale i caratteri vengono visualizzati sullo schermo. Sull'adattatore video monocromatico (MDA) dell'IBM, ogni carattere viene visualizzato all'interno di una matrice di carattere larga 9 punti e alta 14 punti. Sull'adattatore grafico a colori (CGA), la matrice di carattere è 8 per 8.

MCGA: Multi-Color Graphics Array (Matrice grafica integrata multicolore); il sistema di visualizzazione integrato nel modello 30 del PS/2. Anche Memory Controller Gate Array (gate array del controller di memoria) uno dei componenti del sistema di visualizzazione del modello 30.

MDA: Monochrome Display Adapter (adattatore video monocromatico) dell'IBM.

MDPA: Adattatore monocromatico video-stampante; come MDA.

monitor: Dispositivo hardware che visualizza l'output del computer; è composto da un CRT (tubo a raggio catodico) e dalla circuiteria relativa.

MPA: Monochrome/Printer Adapter (adattatore monocromatico/stampante); come MDA.

monitor attivo: In un computer che contiene due sottosistemi di visualizzazione e monitor, il monitor al quale il programma invia il proprio output.

pagina di codice: Un set di caratteri progettato per l'uso con i computer. Ogni carattere di una pagina di codice è associato ad un codice numerico (come ad esempio un codice ASCII o un codice EBCDIC).

PGA: Professional Graphics Adapter (adattatore grafico professionale); un altro nome del PGC IBM.

PGC: Professional Graphics Controller (controllore grafico professionale) dell'IBM.

piano di bit: RAM del video contenente dati grafici formattati. Nei sistemi di visualizzazione IBM possono essere indirizzati in parallelo fino a quattro piani di bit, con valori di pixel rappresentati dai bit alle locazioni corrispondenti nei piani di bit.

pixel: Un punto in un'immagine che è composta da una matrice di punti. L'immagine sullo schermo o su una pagina stampata da una stampante a matrice di punti è composta da moltissimi pixel.

PS/2: Personal System/2.

rapporto dell'aspetto: Il rapporto tra la larghezza di uno schermo e la sua altezza. Un tipico schermo di PC IBM ha un rapporto di aspetto di circa 4:3. Questo termine vie-

ne anche frequentemente utilizzato per descrivere i pixel: se pensate ad un pixel come se fosse rettangolare, il suo rapporto di aspetto sarà il rapporto tra la sua larghezza e la sua altezza.

raster: Gruppo di linee di scansione orizzontali ravvicinate che costituiscono un'immagine video.

RGB: Red, Green, Blue (rosso, verde, blu); i tre colori primari visualizzati dai monitor usati con i sistemi di visualizzazione dei PC e dei PS/2. Tutti gli altri colori sono mescolanze di questi tre colori primari. I monitor controllati da segnali separati di rosso, verde e blu vengono spesso denominati monitor RGB.

set di caratteri: Una serie di caratteri alfabetici e numerici e di simboli.

tavolozza: Gamma dei colori che possono essere visualizzati da un sistema di visualizzazione.

tipo di caratteri: Una descrizione dello stile e della forma dei caratteri in un set di caratteri.

VDI: Interfaccia grafica di un dispositivo virtuale per computer; una proposta di standrd ANSI per un'interfaccia grafica ad alto livello. Il Graphics Development Toolkit (GDT) venduto dall'IBM e dalla Graphics Software Systems è un'implementazione commerciale della VDI.

VGA: Video Graphics Array (Matrice Video Grafica Integrata). Gli utenti si riferiscono al sistema di visualizzazione integrato nei modelli 50, 60 e 80 del PS/2 e all'adattatore di visualizzazione del PS/2 IBM, come "VGA". Tuttavia, la VGA è la circuiteria del sistema di visualizzazione che svolge i compiti del controller del CRT, del sequencer, del controller grafico e del controller di attributo. La maggior parte della circuiteria è contenuta in un singolo chip VLSI.

VLSI: Very Large Scale Integration (integrazione a larghissima scala).

GRUPPO EDITORIALE JACKSON
Cod. R780

Finito di stampare nel mese di Aprile 1989
da Grafica '85 - Rodano Millepini (MI)

PER IL PROGRAMMATORE MS-DOS E PS/2

SCHEDE GRAFICHE

TECNICHE AVANZATE DI PROGRAMMAZIONE

RICHARD WILTON

Il programmatore esperto troverà in questo libro tutte le informazioni necessarie per sfruttare le stupefacenti capacità di visualizzazione dei PC e dei PS/2 e produrre grafica video veloce, professionale e anche di grande impatto visivo. Indipendentemente dalla configurazione hardware in possesso (EGA, CGA, HGC, MDA, VGA o MCGA), l'utente troverà consigli ed esempi specialistici e completi che lo aiuteranno ad affrontare i difficili problemi della programmazione video.
L'autore, Richard Wilton, esamina in dettaglio l'hardware di visualizzazione dei PC e dei PS/2 e fornisce una rivisitazione delle routine di supporto video del BIOS ROM. Egli tratta anche i principi fondamentali dell'architettura hardware, delle modalità di visualizzazione e dell'interfaccia programma/macchina, argomenti che costituiscono i prerequisiti per produrre dell'efficace programmazione video. Il nucleo del libro comprende tecniche comprovate e approfondimenti preziosi, corredati da innumerevoli esempi in codice sorgente, per la creazione di svariati output di testo e di grafica.

Sommario
- Tracciamento di linee
- Cerchi ed ellissi
- Riempimento di aree
- Testo grafico
- Set di caratteri alfanumerici
- Blocchi di bit
- Animazione
- Interrupt di ritracciamento verticale
- Piani di bit
- Programmi ad alto livello e driver di visualizzazione
- Utilizzo dei pacchetti di output video commerciali

GRUPPO EDITORIALE JACKSON

Cod. R780

779

RICHARD WILTON

SCHEDE GRAFICHE

TECNICHE AVANZATE DI PROGRAMMAZIONE